(4)

Center for the Study of Learning

LEARNING RESEARCH AND DEVELOPMENT CENTER

University of Pittsburgh

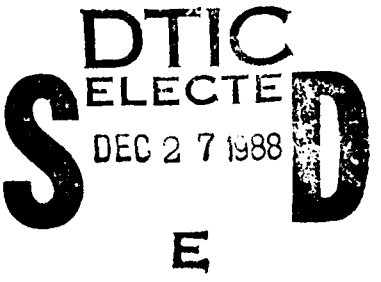
4

An Information Processing Analysis of the Function of Conceptual Understanding in the Learning of Arithmetic Procedures

Stellan Ohlsson and Ernest Rees

The Center for the Study of Learning
at the Learning Research and Development Center,
University of Pittsburgh
Pittsburgh, Pennsylvania 15260

Technical Report No. KUL-88-03
August, 1988



Preparation of this manuscript was supported by ONR grant N00014-85-K-0337, by the OERI institutional grant for The Center for the Study of Learning, and by the Xerox University Grant to the University of Pittsburgh. The opinions expressed do not necessarily reflect the position of the sponsoring agencies, and no endorsement should be inferred.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION: Unclassified	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION / AVAILABILITY OF REPORT: Approval for public release, distribution unlimited
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S): UPITT/LRDC/ONR/KUL-88-03	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION: Learning Research & Development Center, University of Pittsburgh; 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION: Cognitive Science Program, Office of Naval Research (Code 1142CS)
6c. ADDRESS (City, State, and ZIP Code): 3939 O'Hara Street, Pittsburgh, PA 15260	7b. ADDRESS (City, State, and ZIP Code): 800 North Quincy Street, Arlington, VA 22217-5000
8a. NAME OF FUNDING / SPONSORING ORGANIZATION; 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER: N00014-85-K-0337
8c. ADDRESS (City, State, and ZIP Code)	10. SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT ACCESSION NO.
61153N	RR04206	RR040206-0A	NR442a-54

11. TITLE (Include Security Classification)
AN INFORMATION PROCESSING ANALYSIS OF THE FUNCTION OF CONCEPTUAL UNDERSTANDING IN THE LEARNING OF ARITHMETIC PROCEDURES (UNCLASSIFIED)

12. PERSONAL AUTHOR(S): Stellan Ohlsson and Ernest Rees

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Technical	FROM _____ TO _____	1988, August	

16 SUPPLEMENTARY NOTATION
Partial funding by OERI institutional grant for The Center for the Study of Learning, and the Xerox University Grant to the University of Pittsburgh

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Arithmetic, Computer simulation, Constraints, Declarative knowledge, Machine learning, Mathematics education
05	10		
23	02		

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

School children learn arithmetic procedures by rote, rather than by constructing them on the basis of their understanding of numbers. Rote learning produces lack of flexibility, nonsensical errors, and other difficulties in learning. Mathematics educators have proposed that if arithmetic procedures were constructed under the influence of conceptual understanding of the principles of arithmetic, then procedure acquisition would not suffer from these difficulties. However, little effort has been investigated in conceptual analysis of this hypothesis, or in proving its viability. We propose a theory of conceptual understanding and its role in the learning and execution of arithmetic procedures. The basic hypothesis of the theory is that principles constrain the possible states of affairs, and thereby enable the learner to monitor his/her own performance and to correct his/her errors. We propose a new knowledge representation, the state constraint, which captures this view of principled knowledge.
(Continued on back)

20 DISTRIBUTION / AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	Unclassified
22a NAME OF RESPONSIBLE INDIVIDUAL: Susan M. Chipman	22b. TELEPHONE (Include Area Code): 202-696-4318; 22c. OFFICE SYMBOL: ONR 1142pt

DD Form 1473, JUN 86 — Previous editions are obsolete. — SECURITY CLASSIFICATION OF THIS PAGE

UNCLASSIFIED

19. ABSTRACT

(Continued from Report Documentation Page, Block 19)

The state constraint theory has been implemented in the Heuristic Searcher (HS), a computer model that learns arithmetic procedures on the basis of general principles encoded as constraints on search states. We have simulated (a) the discovery of a correct and general counting procedure in the absence of either instruction or solved examples, (b) flexible adaptation of an already learned counting procedure in response to changes in the task demands, and (c) the correction of errors in multi-column subtraction in the absence of external feedback. The state constraint theory provides novel answers to several questions with respect to conceptual understanding in arithmetic, generates counter-intuitive but testable predictions about human behavior, deals successfully with technical issues that cause difficulties for other explanations of the function of knowledge in learning, and fares well on evaluation criteria such as generality and parsimony. The state constraint theory is incomplete; it does not explain how procedure acquisition proceeds in the absence of conceptual understanding, or how learners overcome errors that can not be described as violations of principles. Future work will focus on the question of how knowledge and experience interact in procedural learning.

Knowledge and Understanding in Human Learning

Knowledge and Understanding in Human Learning (KUL) is an umbrella term for a loosely connected set of activities lead by Stellan Ohlsson at the Learning Research and Development Center, University of Pittsburgh. The aim of KUL is to clarify the role of *world knowledge* in human thinking, reasoning, and problem solving. World knowledge consists of general principles, and contrasts with facts (episodic knowledge) and with cognitive skills (procedural knowledge). The long-term goal is to answer four questions: How are new principles acquired? How are principles utilized in insightful performance? How are principles utilized in learning to perform? How can instruction facilitate the acquisition and utilization of principled (as opposed to episodic or procedural) knowledge? Different methodologies are used to investigate these questions: Psychological experiments, computer simulation, historical studies, semantic, logical, and mathematical analyses, instructional intervention studies, etc. A list of KUL reports appear at the back of this report.

Table of Contents

Abstract

School children learn arithmetic procedures by rote, rather than by constructing them on the basis of their understanding of numbers. Rote learning produces lack of flexibility, nonsensical errors, and other difficulties in learning. Mathematics educators have proposed that if arithmetic procedures were constructed under the influence of conceptual understanding of the principles of arithmetic, then procedure acquisition would not suffer from these difficulties. However, little effort has been investigated in conceptual analysis of this hypothesis, or in proving its viability. We propose a theory of conceptual understanding and its role in the learning and execution of arithmetic procedures. The basic hypothesis of the theory is that principles constrain the possible states of affairs, and thereby enable the learner to monitor his/her own performance and to correct his/her errors. We propose a new knowledge representation, the *state constraint*, which captures this view of principled knowledge. The state constraint theory has been implemented in the Heuristic Searcher (HS), a computer model that learns arithmetic procedures on the basis of general principles encoded as constraints on search states. We have simulated (a) the discovery of a correct and general counting procedure in the absence of either instruction or solved examples, (b) flexible adaptation of an already learned counting procedure in response to changes in the task demands, and (c) the correction of errors in multi-column subtraction in the absence of external feedback. The state constraint theory provides novel answers to several questions with respect to conceptual understanding in arithmetic, generates counter-intuitive but testable predictions about human behavior, deals successfully with technical issues that cause difficulties for other explanations of the function of knowledge in learning, and fares well on evaluation criteria such as generality and parsimony. The state constraint theory is incomplete; it does not explain how procedure acquisition proceeds in the *absence* of conceptual understanding, or how learners overcome errors that can *not* be described as violations of principles. Future work will focus on the question of how knowledge and experience interact in procedural learning.

Rote vs. Meaningful Learning in Arithmetic

School children tend to learn arithmetic procedures by memorizing them, rather than by constructing them on the basis of their understanding of numbers. Consequently, they execute those procedures mechanically, as sequences of physical actions on written characters rather than as abstract operations on numbers. If they arrive at correct answers, it is because they recall the relevant procedure accurately, not because they understand the underlying mathematical concepts and principles.

Rote learning of arithmetic procedures has several negative consequences. Memorized procedures are *brittle*. They lack the flexibility required to transfer to unfamiliar problems or even to minor variations of familiar problems. Students often fail on a novel task that is conceptually equivalent to, but procedurally distinct from, some other, already mastered task. Inability to adapt a procedure to changes in the task implies that each new task has to be learned separately.

Memorized procedures are also prone to *nonsensical errors*. For instance, in the so-called SMALLER-FROM-LARGER error in multi-column subtraction (Brown & Burton, 1978; Burton, 1982), the student subtracts the smaller number from the larger within each column without regard for which number belongs to the minuend and which number belongs to the subtrahend. In the so-called FRESHMAN error (Silver, 1986, p. 189) in the addition of fractions the learner adds the denominators as well as the numerators, and in what we might call the DECIMAL-AS-INTEGER error, the learner judges the relative size of decimal fractions on the basis of their integer values (Hiebert & Wearne, 1986, p. 205). These errors are nonsensical because they violate the meaning of the corresponding arithmetic operations. Nonsensical errors slow down learning because they resist remedial instruction.

Finally, memorized procedures *resist being incorporated as subprocedures into higher-order procedures*. Students often fail to perform steps *A* and *B* in combination, even though they are capable of performing both *A* and *B* in isolation. For instance, we have observed in our field studies children who know how to put two fractions on the same denominator and who also know how to add two fractions with equal denominators, but who nevertheless are unable to figure out how to add two fractions with unequal denominators.[1] Since mathematics is a hierarchically organized subject matter, inability to build on previously mastered procedures severely limits the mathematics that can be learned.

The working hypothesis that dominates current research in mathematics education is that conceptual understanding is the cure for these negative effects. We will refer to this belief as the *Conceptual Understanding Hypothesis*. If children understood what they are doing, this hypothesis claims, children could discover procedures on their own, learned procedures would be flexible, nonsensical errors would be corrected spontaneously (or at least not be persistent to remediation), and already mastered procedures would easily combine to form higher-order procedures. The Conceptual Understanding Hypothesis claims that procedures can be derived from the learner's knowledge, in contrast to being

[1] Our empirical research on the learning of fractions will be reported elsewhere.

derived either from experience or from an external source such as a teacher or a textbook. In previous work we called this type of learning *rational learning* (Ohlsson, 1986, 1987b; Ohlsson & Rees, 1987). The Conceptual Understanding Hypothesis extends the idea of rational learning by claiming that procedures which are derived from knowledge are more flexible and less error-prone than procedures that are learned in other ways.

Common sense strongly supports the Conceptual Understanding Hypothesis, but, as Brooks and Dansereau (1987, pp. 134-136) point out in their recent review of what they call content-to-skill transfer, it has been the subject of a surprisingly small amount of systematic research. There are scattered studies that demonstrate a facilitating effect of understanding a principle on subsequent problem solving (Egan & Greeno, 1973; Mayer, Stiehl, & Greeno, 1975; Katona, 1967). However, the strongest case for conceptually based procedure acquisition has been made by Gelman and co-workers with respect to counting (Gelman & Gallistel, 1978; Gelman & Meck, 1983, 1987; Gelman, Meck, & Merkin, 1986; Greeno, Riley, & Gelman, 1984). Gelman and Gallistel (1978) formulated a set of principles that determine the correct procedure for counting. The three most important are the One-One Mapping Principle, the Cardinal Principle, and the Stable Order Principle. The *One-One Mapping Principle* states that each object should be assigned exactly one number. The *Cardinal Principle* states that the last number to be assigned to an object is also the answer to the counting problem. The *Stable Order Principle* states that the numbers have to be considered in numerical order. Gelman and co-workers have presented evidence for the hypothesis that children know these principles before they have a procedure that enables them to count correctly, and that they construct their counting procedures on the basis of these principles. The evidence includes the facts that children typically acquire the correct procedure for counting without formal instruction in counting, and that their counting procedures are flexible. Children readily adapt their procedures for counting to non-standard counting tasks, such as counting objects in a particular order, or in such an order that a specified object is assigned a specified number (Gelman & Gallistel, 1978). Greeno, Riley, and Gelman (1984) and Smith, Greeno, and Vitolo (in press) have proposed a theoretical analysis that shows how flexible counting performance can be derived by a planning mechanism from a set of action schemata that embody the counting principles, thus lending support to this interpretation of the evidence. In short, research suggests that the normal acquisition of counting in our culture exemplifies the Conceptual Understanding Hypothesis.[2]

If counting represents a clear example of knowledge-based procedure acquisition in arithmetic, then

[2]The conclusion that children know the counting principles before they learn counting procedures is not uncontested. Piaget (1952) concluded on the basis of his research that children do not understand number in the pre-operational stages, because the construction of number is coordinated with the construction of logical operations. Brainard (1979) has argued on the basis of extensive empirical studies that the notion of ordinality develops before the notion of cardinality, a conclusion which complicates the relation between counting and the Cardinality Principle. Both Fuson & Hall (1983) and Briars and Siegler (1984) have proposed accounts of childrens' counting that assume that procedures are learned before principles. Baroody & Ginsburg (1986, pp. 76-78) agree with this view. This view is further supported by recent studies by Douglas Frye, Nicholas Braisby, John Lowe, Celine Maroudas, and Jon Nicholls at the University of Cambridge, England (personal communication). Since our purpose in this report is to present a computational interpretation of the Conceptual Understanding Hypothesis, rather than to make a critical appraisal of the empirical literature, we have adopted the *principles first* view as our working hypothesis. Clearly, the Conceptual Understanding Hypothesis retains its interest as a pedagogical stance, even if the debate about childrens' counting should ultimately be resolved in favour of the *procedures first* view.

multi-column subtraction represents the opposite. The evidence for rote learning is particularly strong with respect to this procedure. Over one hundred distinct error types have been identified in childrens' subtraction performances, most of them as nonsensical as the prototypical SMALLER-FROM-LARGER error mentioned above (Brown & Burton, 1978; Burton, 1982; Young & O'Shea, 1981). Kurt VanLehn has proposed a theory that assumes that understanding of, say, place value does not enter into the acquisition of the procedure for multi-column subtraction as it actually occurs in the classroom (Brown & VanLehn, 1980, 1982; VanLehn, 1983a, 1983b, 1985a, 1985b, 1986). According to his theory, children pay little attention to, or are intellectually unequipped to make much use of, teachers' explanations of the subtraction procedure. Instead, they construct the procedure by induction over the solved examples provided by textbooks and teachers. If the resulting procedure is incomplete, the learner may encounter situations in which the procedure cannot be executed, so-called *impasses*. The learner is hypothesized to respond to such difficulties by making local changes in the procedure. VanLehn's theory explains a significant proportion of the empirically observed procedural errors for multi-column subtraction, thus strongly supporting the notion that children learn the subtraction procedure by rote.

In summary, research has provided us with in-depth analyses of two contrasting examples of procedure acquisition in arithmetic. The case of counting exemplifies procedure acquisition based on understanding of the relevant principles, and the case of subtraction exemplifies procedure acquisition through memorization. The subtraction research is silent on the question of whether conceptual understanding *could* facilitate the learning of subtraction. It only makes the case that the acquisition of the subtraction procedure as it currently occurs in schools does not, in fact, engage the learner in the mathematics that underlies that procedure. The pedagogical hope expressed in the Conceptual Understanding Hypothesis is that the subtraction procedure *could* be acquired in the same intelligent manner as the counting procedure, if only children understood the principles of subtraction as well as they understand the principles of counting.

The obvious instructional implication of the Conceptual Understanding Hypothesis is that we need to find ways of teaching children to understand the conceptual underpinnings of arithmetic procedures. A significant proportion of research in mathematics education is directed towards this goal (see, e. g., Bell, Costello, & Kuchemann, 1983; Davis, 1984; Hiebert, 1986; Romberg & Carpenter, 1986; Shoenfeld, 1985; Silver, 1985).

The research reported here has a different purpose. Our goal is to clarify the nature of the hypothesized link between conceptual understanding and procedure acquisition. *How* does conceptual understanding facilitate procedure acquisition? In a major review of the psychology of mathematics Resnick and Ford (1981) summarized the state of the research with respect to this questions as follows:

> The relationship between computational skill and mathematical understanding is one of the oldest concerns in the psychology of mathematics. It is also one that has consistently eluded successful formulation as a research question. ... Instead of focusing on the *interaction* between computation and understanding, between practice and insight, psychologists and mathematics educators have been busy trying to demonstrate the superiority of one over the other. ... The *relationships* between skill and understanding were never effectively elucidated. What is needed, and what now seems a possible research agenda, is to focus on *how* understanding influences the aquisition of computational routines ... .
>
> (Resnick & Ford, 1981, p. 246)

Information processing analyses of human cognition imply that an analysis of the relation between conceptual understanding and performance consists of two components: A representation for conceptual understanding plus a computational machinery that can derive a procedure for a particular task from that understanding (Greeno, Riley, & Gelman, 1984; Smith, Greeno, & Vitolo, in press). Such an analysis should explain how conceptual understanding is represented in memory, how it functions in performance, and how it can facilitate learning. The work reported here is based on this formulation of the problem.

We approach this problem by building a computer model of learning that instantiates the Conceptual Understanding Hypothesis. Such a model has many uses. First, the model can provide what is known as a *sufficiency proof* (Newell and Simon, 1959, p. 5). The model can provide a concrete demonstration that the kind of learning that mathematics educators envision is, in fact, possible. Second, the model can serve as a tool for generating predictions from a particular set of hypotheses about understanding. Third, it can serve as a focus of debate. Other researchers may not agree that our model represents learning as it actually occurs in, say, the case of counting, or as it ought to proceed in the classroom. The formulation of alternative interpretations of the Conceptual Understanding Hypothesis ought to be facilitated by having something precise to disagree with. Fourth, our model can serve as a tool for the planning of empirical studies of the role of conceptual understanding in the learning of procedures. Fifth, it can be the basis of diagnostic instruments that focus on misconceptions rather than on bugs (Langley, Wogulis, & Ohlsson, in press). Sixth, it can facilitate comparison between the Conceptual Understanding Hypothesis and other hypotheses being explored in current research on learning. Seventh, it can be used to derive instructional implications that can be tested in classroom interventions.

The report is organized as follows. We begin by stating a theory of conceptual understanding and its relation to performance and to procedure acquisition (*The State Constraint Theory of Understanding*, p. 8). In the second section we describe a computer model based on this theory (*A Computer Model*, p. 19), and in the following section (*Computational Results*, p. 28) we report on three applications of the model: (a) the construction of a counting procedure in the absence of explicit instruction or solved examples, (b) the adaptation of an existing counting procedure to changes in the counting task, and (c) the spontaneous correction of procedural errors in multi-column subtraction. We then compare our work with previous efforts to simulate procedure acquisition in arithmetic (*Relation to Previous Research*, p. 60), and discuss its implications (*General Discussion*, p. 69).

The State Constraint Theory of Understanding

A theory of the role of conceptual understanding in the acquisition of procedures consists, at the broadest level of analysis, of two components: a representation for conceptual understanding and a computational machinery that maps that understanding onto a procedure for a particular task (Greeno, Riley, & Gelman, 1984; Smith, Greeno, & Vitolo, in press): More specifically, such a theory should answer at least the following questions:

1. What is the nature of conceptual understanding, and how is it represented in the mind? What kind of cognitive structures are we referring to when we speak of someone as understanding, say, multi-column subtraction?

2. What function does conceptual understanding have in *performance*? How does understanding interact with the procedure during execution? What is the difference between executing a procedure correctly and with understanding, as opposed to executing it correctly but without understanding?

3. What function does conceptual understanding have in the *learning* of procedures? By what mechanism does understanding enter into the construction of a procedure? How does understanding enable the learner to discover a procedure, to apply a procedure in a flexible manner, to correct nonsensical errors, and to combine procedures into higher-order procedures?

The theory proposed here is based on the idea that learners act with understanding when they internally monitor their performance on a problem by comparing the successive states of the problem with what they know about the task environment. According to this theory learners execute the procedure for, say, multi-column subtraction with understanding when they think about each state of the subtraction problem in terms of the principles of arithmetic. Learning occurs when an incorrect or incomplete procedure generates a problem state that is inconsistent with the principles that govern the task environment[3]. Cognitive change is in the direction of greater consistency between the learner's actions and the structure of the task environment (to the extent that the latter is known to the learner). For instance, an incorrect subtraction procedure may result in a difference between two integers that is larger than the minuend. To the extent that the learner knows that $n - m = r$ implies $r < n$, the subsequent revision of the regrouping procedure is in the direction of preventing violations of this principle in future applications of that procedure, or so the theory claims.

The purpose of this section is to state our hypotheses about understanding, about performance, and about learning. In the next section we describe a computer model that instantiates these hypotheses (*A Computer Model*, p. 19). In a later section we describe some results obtained by running the model

[3]It may seem as if problem states that violate the principles of the environment are impossible in non-symbolic domains. For instance, one cannot construct, say, an electronic circuit that violates the principles of electricity. However, the term "problem state" as used in our theory refers to the *mental representation* of the state of the problem, not to the physical problem situation. This point will be clarified in the subsection that presents our performance theory (*Hypotheses about performance*, p. 12).

(*Computational Results*, p. 28). The simulation runs show that the hypotheses stated here predict learning behavior that is consistent with the Conceptual Understanding Hypothesis.

Hypotheses about understanding

In this report we use the term "understanding" to refer to a collection of general principles about the environment, formulated as constraints on the possible states of affairs. We unpack this notion in four steps.

Understanding consists of knowledge about the task environment

The Conceptual Understanding Hypothesis claims that correct and flexible performance is achieved when the learner constructs the required procedure on the basis of his/her understanding. The type of understanding that we focus on in this research is *understanding of the domain in which a procedure operates*. To understand a domain is to know the principles that govern the objects and events in that domain. For instance, to understand electricity is to know the principles that govern the behavior of electric currents; to understand arithmetic is to know the laws of numbers. This type of understanding is central to the learning-by-doing scenario, in which the learner constructs a procedure in the absence of instruction.

An alternative view is that to understand a procedure is to know the *purpose* of each step in the procedure. Such an understanding is sometimes called a *teleological semantics* for the procedure (VanLehn & Brown, 1980). A second view of understanding is that one understands *X* when one subsumes *X* under some existing cognitive structure. We might call this *representational* understanding, since it emphasizes the encoding of a problem (as opposed to the procedure for solving it). The subsumption theory of understanding has been applied both to problem solving (Greeno, 1978, 1983; Anderson, Greeno, Kline, & Neves, 1981), and to text understanding (e. g., Galambos, Abelson, & Black, 1986; Schank, 1986). Yet another view is that to understand a procedure is to know the *justification* for the procedure. This conception of understanding is common among professional mathematicians. Both teleological and justificatory understanding are crucial in the learning-by-being-told scenario, in which a teacher demonstrates the execution of a procedure and then explains that procedure, i. e., verbally communicates its teleology and its justification. A complete theory of understanding would specify the nature and function of both conceptual, teleological, representational, and justificatory understanding. Michener (1978) has proposed such a multi-facetted view of mathematical understanding.

Knowledge is declarative rather than procedural

Current cognitive theory recognizes two kinds of knowledge, *declarative knowledge* and *procedural knowledge* (Winograd, 1975). This distinction is essential to the theory proposed here. For instance, consider the assertion that

Uppsala is ninety kilometers north of Stockholm.

This assertion is not a procedure; it does not specify how to accomplish any task. But it is relevant for

many different procedures[4], such as *if you are in Uppsala and your goal is to get to Stockholm, then travel south for ninety kilometers* and *if you are in Stockholm, and your goal is to get to Uppsala, then travel north for ninety kilometers.* The set of procedures for which an assertion is relevant is open-ended. As an example of a less immediate consequence of the above assertion, consider the procedure *if you are midway between Uppsala and Stockholm, and feel like getting as far away from both as possible, then travel either straight west or straight east.* The only limits on the set of procedures for which an assertion is relevant are the limits on our imagination. As a second example, consider the following arithmetic principle:

> *A set of numbers always yield the same sum, regardless of the order in which they are added.*

This principle does not in itself specify how to accomplish any particular task, but the set of procedures for which it is relevant is open-ended. For instance, the above principle is crucial for the standard procedure for multi-column subtraction because it enables regrouping of the minuend.

Declarative and procedural knowledge differ along three dimensions. First, declarative knowledge is *goal-independent*, while procedural knowledge is *goal-related.* Declarative knowledge is knowledge about what the world is like, while procedural knowledge is knowledge about how to attain particular objectives. Declarative knowledge is potentially useful in reaching an infinite range of goals, including goals that the learner had never thought of at the time of storing the knowledge in memory.

Second, declarative knowledge is *context-free* while procedural knowledge is *situated.* Uppsala is *always* ninety kilometers north of Stockholm; the distance is not a function of the current situation of the person who is making use of this fact. But the procedure for getting to Uppsala by travelling ninety kilometers northward is only useful if the person finds himself/herself in Stockholm; it does not lead to the goal if executed in any other situation. Similarly, a sum of a set of numbers is unique; it is not a function of the problem the agent is trying to solve. But the regrouping procedure is appropriate only with respect to subtraction problems in which some minuend digit is larger than the corresponding subtrahend digit.

Third, declarative knowledge is assertive or *descriptive*, while procedural knowledge is exhortational or *imperative.* Declarative knowledge relates objects and events in the world to other objects or events, while procedural knowledge relates situation/goal pairs to actions. Procedural knowledge is knowledge about what to do in in order to obtain some particular state of affairs. It is neither true nor false, but more or less *effective*; executing a certain action in a particular situation will lead to attainment of the relevant goal with more or less expenditure of time, cost, or effort.

In the research reported here we take the stance that the term "procedural knowledge" is, strictly

[4]A procedure typically consists of a (possibly very large) collection of rules. The simple procedures discussed in this section consist of just a single rule each.

speaking, a misnomer.[5] Procedures do not encode knowledge; they encode dispositions to act in particular ways under particular circumstances. Hence, understanding cannot be encoded in action schemata, methods, operations, rules, or other procedural representations. The opposite stance is that all knowledge is procedural. For example, the Soar simulation model by Allen Newell and co-workers (Laird, Rosenbloom, & Newell, 1986) is build on the assumption that all knowledge is encoded in production rules; the Soar system does not have any other representational format. A compromise stance is that knowledge can be either procedural or declarative. For example, the ACT* model (Anderson, 1976, 1983) is build on the assumption that there are separate memories for propositions and for rules.

Understanding consists of principled rather than factual knowledge

Declarative knowledge can be divided into two types. Abstract or *principled* knowledge consists of assertions about universals. The principle that the sum of a set of numbers is unique states something about arithmetic sums in general. *Factual* knowledge, in contrast, consists of assertions about particular objects or events. The statement that Uppsala is ninety kilometers north of Stockholm is an example of factual knowledge. A factual assertion that refers to a particular spatiotemporal context is sometimes classified as an instance of *episodic* knowledge.

Cognitive psychology has produced a wealth of information about the storage, retention, and retrieval of factual, particularly episodic, information. However, the Conceptual Understanding Hypothesis emphasizes principled rather than factual knowledge. The idea that we have explored in the research reported here is that general principles can guide the construction of arithmetic procedures. Factual knowledge is not foreign to arithmetic--for instance, *three is an odd number* is a factual assertion--but it is less relevant for our current purpose than principled knowledge.

There are severe philosophical problems associated with the concept of principled knowledge. For instance, since abstract properties of the world are not directly perceivable, the question arises how we can have knowledge about them. Furthermore, since a general principle is not tied to a particular spatiotemporal context, it is not clear what it means for such a principle to be either true or false. A significant proportion of research in epistemology is devoted to clarifying these problems. However, the research we report here does not presuppose solutions to the problems of philosophy. We are investigating the psychological question of how the principles a student believes can guide his/her procedure acquisition; we are not trying to decide whether he/she is justified in believing those principles.

The alternative hypothesis is that declarative knowledge consists mainly or exclusively of factual knowledge. This hypothesis has the advantage of avoiding the philosophical problems associated with principled knowledge. But we do not perceive a need to argue for the existence or the psychological reality of principled knowledge as a *preliminary* to the research reported here. On the contrary, we expect

[5]Since the use of the term "procedural knowledge" to refer to dispositions to act is so widespread, we will adhere to that usage throughout this report.

a conclusion about the usefulness of the concept of principled knowledge to be one of the *outcomes* of our research.

Principles constrain the possible states of affairs

Traditional debates about the nature of knowledge assume that knowledge consists either of descriptions ("All swans are white") or predictions ("The sun will rise tomorrow"). In this report we focus on a different aspect of principled knowledge. We view principles as *constraints* on the possible states of the world. An obvious example is the following common sense principle:

Two objects cannot occupy the same space at the same time.

As a descriptive statement, this principle is not very informative; it does not tell us much about what the world is like.[6] Nor is it predictive; it does not by itself assert that such and such an event will happen.[7] The impact of the above principle is to rule out certain states of affairs as impossible; it claims that situations in which two material objects occupy the same physical space will not occur. Many physical laws, e. g. laws of conservation, have the character of constraints (Feynman, 1965).

The notion of principled knowledge as consisting of constraints on the possible states of affairs is particularly relevant for arithmetic. Arithmetic principles, e. g., the principles of commutativity and associativity, do not predict which arithmetic operations will occur. Instead, they classify states of affairs into mathematically valid and invalid states, as it were. For instance, the principle of commutativity of addition claims that it cannot happen that we add two numbers in two different orders and get two different answers.

An alternative hypothesis is that principled knowledge consists mainly of predictive principles (Hollan, Holyoak, Nisbett, & Thagard, 1986). We are not claiming that all principles can be formulated as constraints. We would expect an exhaustive investigation into principled knowledge to reveal many different kinds of principles. We do claim that constraints are frequent and particularly important in arithmetic, a domain in which other types of principles, particularly predictive principles, are not relevant.

Hypotheses about performance

Learning is a change in performance. Hence, specific hypotheses about learning presupposes specific hypotheses about the nature of performance. The purpose of this subsection is to state our hypotheses about the cognitive machinery that executes a procedure, and about the function of principled knowledge in such execution.

---

[6] It contrasts in this regard with a principle like *planets travel in elliptical orbits*, which does have descriptive content.

[7] It contrasts in this regard with a principle like the traditional Swedish saying that *if the roenneberries turn bright red in the fall, the winter will be very cold*, which does have predictive content. (That fact that an assertion has predictive *content* obviously does not imply that it also has predictive *accuracy*.)

Thinking is heuristic search

We have chosen to work within the theory of thinking proposed by Newell and Simon (1972). The basic idea of their theory is that humans think by searching a problem space. A problem space is defined by (a) the initial state of the problem, (b) the ensemble of operators available for processing the problem, and (c) the criterion for what counts as a goal state. Searching such a space means tentatively applying operators to states in order to find a sequence of operators that lead from the initial state to the goal state. The search is guided by heuristics, rules of the general form *when trying to obtain goal G, and the current situation have properties* P_1, P_2, ... , P_n, *then consider action A.* The reader is referred to the original statement of the theory for details (Newell & Simon, 1972).

There are several reasons for selecting the theory of heuristic search as our performance theory. First, we prefer building on previous research over inventing computational mechanisms *ad hoc* to suit our current purpose. By choosing the main performance theory to emerge in recent research on thinking, we integrate our efforts with other research efforts. Second, the theory of heuristic search is a general theory. The mechanism of heuristic search is applicable to many task domains, not just to arithmetic. By using a computational mechanism that has been applied to a wide range of tasks we increase the plausibility of our theory. Third, the theory of heuristic search is precise enough to guide the construction of a simulation model. Fourth, the theory of heuristic search is better grounded in psychological data than any other current theory of human thinking. It has been used to explain why some problems are more difficult than others (e. g., Kotovsky, Hayes, & Simon, 1985), why people perform differently on a particular problem (e. g., Newell & Simon, 1972, Chaps. 7, 10, and 13), how procedures can be learned from practice (e. g., Anzai & Simon, 1979), etc. In short, there is no other theory with comparable generality, conceptual precision, and empirical grounding.

A further reason to select the hypothesis of heuristic search as our performance theory is that it satisfies the following criterion of adequacy:

> *Criterion of Executability of Partial Procedures. Since procedure learning is gradual, the performance theory underlying a learning theory must enable a procedure to be executable at each stage during its construction.*

A cognitive procedure is not learned in an all-or-none fashion. Rather, the student learns some part of the procedure, flounders, learns some more parts, makes mistates, etc., in a gradual progression through different stages of competence until the procedure is completed.[8] But at each moment in time during this gradual construction the learner is capable of acting, of executing the procedure as it exists at that point in time. This observation constrains the possible theories about the human performance system to those which enable procedures to be executable at each stage of completeness.

[8] At this point, further practice may lead to the discovery of short-cuts, memorization of special cases, elimination of redundancies, chunking of steps that always follow each other, etc. In the research reported here we are concerned with the initial construction of a procedure, rather than with its subsequent automatization.

The hypothesis of heuristic search satisfies the Criterion of Executability of Partial Procedures, because the function of knowledge, according to this hypothesis, is to constrain search, and search can be constrained to a higher or lesser degree. At the most constrained end of the scale the search follows a single, unbranching path through the problem space. To an external observer the resulting behavior looks algorithmic. At the other extreme, the problem space is searched by randomly selecting operators. To an outside observer the resulting behavior looks like aimless floundering. A typical performance during procedural learning is located somewhere between these extremes: The learner knows something about how to search the relevant space, but not everything; hence, he/she proceeds in the general direction of the solution, but makes mistakes along the way. A collection of search heuristics is always executable, regardless of how incompletely it represents the target procedure. The resulting behavior might be ineffectual, but it will be task oriented.

An alternative hypothesis to heuristic search is what we might call *the problem reduction theory*, following the classification by Nilsson (1971) of problem solving methods into search methods and problem reduction methods. The problem reduction theory says that a procedure consists of a hierarchy of goals and subgoals. Each goal acts like a *procedure call* in an applicative programming language like (pure) LISP. A call to a procedure (goal) is executed by calling its subprocedures (subgoals), which leads to calls to its subprocedures (subgoals), etc., until the procedure called is a primitive operator that can be executed without further reduction. In order for the problem reduction theory to satisfy the Criterion of Executability of Partial Procedures, it must be augmented with an hypothesis about what happens when a procedure call cannot be executed. The theory of repairs proposed by Brown and VanLehn (1980, 1982) is such an hypothesis. Repair theory says that when a problem solver cannot execute a procedure call, he/she edits the current control structure for the execution of that procedure in such a way that the problematic procedure call is eliminated; normal execution then resumes.

Principles constrain search through state evaluation

Given the choice of heuristic search as our performance theory, and given our focus on principled knowledge, the research problem we have posed can be re-stated as follows:

> *What role can principled knowledge play in a heuristic search system? How can principled knowledge improve performance and facilitate the revision of search heuristics?*

Heuristic search consists of the execution of actions in the pursuit of some goal in a particular context; where do principles, context-free knowledge items that do *not* relate to goals and that do *not* mention actions, impinge on that process? The hypothesis of heuristic search suggests two different functions for knowledge: Knowledge can enter into the *generation* of search steps and/or it can enter into the *evaluation* of search steps. In accordance with our decision to view principles as constraints, we focus on the evaluative function. We envision principled knowledge as a device for internal self-monitoring of performance. Since this is the central notion of our theory, we will expand it briefly here; more technical details are provided in the section on the performance mechanism of our simulation model (*The performance mechanism*, p. 19).

We hypothesize that principles are encoded in memory as *state constraints*, criteria which a search state has to satisfy in order to be valid or correct. A heuristic search mechanism can compare each search state with those constraints, and decide whether it satisfies the constraints. States that violate one or more constraints are inconsistent with the system's knowledge and should be avoided; they are the results of incomplete or incorrect procedural knowledge. The collection of state constraints thus constitutes a knowledge-based evaluation mechanism that enables the search system to monitor the performance of its own procedural knowledge. For instance, an incomplete or incorrect arithmetic procedure is likely to generate states of affairs that are not in accord with the laws of the number system. A counting procedure that does not select a new object before generating the next number counts the same object repeatedly, thereby violating the constraint that each objects should be associated with exactly one number. A regrouping procedure that performs a decrement without performing the corresponding increment will change the value of the number being regrouped, thereby violating the constraint that the value of the minuend should remain constant during subtraction. State constraints enable a performance mechanism to catch itself, as it were, in making errors.

The hypothesis that the function of principled knowledge is to evaluate search states satisfies the Criterion of Executability of Partial Procedures. The search procedure may be more or less effective, but at each level of effectiveness it is possible to classify the search states generated as either consistent with the available constraints or as violating them. If the search procedure is nearly complete and correct, then there will be few states that violate the system's constraints; if is radically incomplete or incorrect, then many search states will cause constraint violations. But the system is executable regardless of the level of completeness of its procedural knowledge. Principled knowledge can also be more or less complete. If the system knows many constraints, then a large proportion of the invalid states will be caught, as it were. If the system knows only a few of the relevant constraints, then invalid states will slip through, possibly resulting in a wrong answer. But the computational mechanism does not cease to function in the presence of incomplete knowledge.

The alternative hypothesis is that principled knowledge impinges on heuristic search in the generation, rather than in the evaluation, of search steps. This hypothesis is intuitively plausible, and it is implicitly presupposed in many analyses of human thinking, e. g., in analyses of scientific problem solving (e. g., Jones & Langley, 1988), medical reasoning (e. g., Patel & Groen, 1986), etc. There is no reason to expect knowledge to have a single function in thinking and learning. Human beings obviously use knowledge both in generating ideas about what to do and in evaluating the outcomes of their actions. A complete cognitive theory must explain both the generative and the evaluative functions of principled knowledge.

Hypotheses about learning

A theory of learning has two questions to answer. First, *when* does cognitive change occur? When will the performance machinery roll on unchanged, and when will it undergo revision? Second, *what* change will occur? Given the mental conditions that trigger learning, which knowledge structure will be revised, and how will it be revised?

Constraint violations trigger procedure revision

What events trigger the internal change mechanisms? Given a heuristic search system which is equipped with a collection of state constraints and which can monitor its own performance by comparing search states with those constraints, it is natural to assume that learning is triggered by constraint violations. The constraints--the principled knowledge of the system--enables the system to know that its procedure is incorrect and that revisions are needed. If the search procedure is correct and complete, it should never generate a state that violates any constraint. A constraint violation indicates that the procedure is faulty, and should be revised in such a way that application of that procedure in the future will not lead to further constraint violations.

Many alternative hypotheses about the mental conditions that trigger learning are possible. Some learning theories assume that learning is continuous. For instance, Neves and Anderson (1981, p. 73) investigated the assumption that whenever two procedural rules are applied in sequence, the procedure is extended with the composition of those two rules. Traditional S-R theories (Neimark & Estes, 1967) as well as connectionist learning theories (Hinton, 1987) also assume that learning happens on every trial. Other learning theories tie learning to the goal structure of the procedure being executed. For instance, the UPL model (Ohlsson, 1983a, 1987a) and the Soar model (Laird, Rosenbloom, & Newell, 1986) both learn when they succeed in satisfying a subgoal. A different triggering criterion was proposed by Neches (1981, 1982, 1987). His model of heuristic procedure modification is based on the assumption that learning is triggered by the discovery of patterns in the internal trace of a procedure, patterns that indicate that there are redundancies in the procedure that can be eliminated. The formulation of the triggering condition for a particular theory obviously depends on the knowledge structures postulated by that theory.

Constraint violations inform procedure revisions

Given that the current search procedure has generated a search state that violates a constraint, what change should occur? We postulate that a constraint violation not only signals that a revision is needed, but also that it contains information about how the faulty procedure should be revised. We propose that the required change can be derived from the system's knowledge. We have called this idea the *Rational Learning Hypothesis* in previous work (Ohlsson, 1987b; Ohlsson & Rees, 1987), because it claims that the learning mechanism has rational grounds for the change that it brings about.

The learning mechanism of our simulation model can identify the circumstances that lead to a constraint violation, and revise the relevant rule in the appropriate way. A precise statement of the algorithm that accomplishes this will be given in the next section. The basic idea is as follows. Suppose that state S_1 is consistent with all available state constraints, but that operation A transforms S_1 into state S_2, which does violate a constraint C. The cause of the violation is then to be found in the changes A caused in S_1. By looking at the those changes, and relating them to the violation, we can pinpoint the reason why executing A in S_1 lead to the violation of a constraint. The rule that applied A can then be revised in such a way that it recognizes situations in which A will have the effect of violating that constraint, and avoids executing A in those situations.

The hypothesis of rational learning contrasts with the two alternative hypotheses of learning by *induction* and learning by *analogy*. The dominant hypothesis in current learning theory is that new cognitive structures are constructed by identification of the commonalities of a set of examples. When the examples are successful problem solving steps, the inductive hypothesis becomes a theory of learning through practice. A number of variations on this theme have been explored (see the collections of articles edited by Anderson, 1981; by Bolc, 1987; and by Klahr, Langley, & Neches, 1987). Another alternative hypothesis is that humans learn primarily from factual or episodic knowledge. The solution to a novel problem is hypothesized to be constructed by remembering the solution to some previously solved problem, which is then edited, as it were, to fit the new problem. The hypothesis of learning by analogy has been explored by a number of researchers (Adelson, Gentner, Hammond, Holyoak, & Thagard, 1988; Carbonell, 1982, 1983; Gentner, 1987; Holyoak, 1984; Rumelhart & Norman, 1981). Human beings are also capable of learning by *being told* (Hayes-Roth, Klahr, & Mostow, 1981). Both inductive learning, analogical learning, and learning by being told are important psychological processes that will have to be included in a complete theory of learning.

Summary of hypotheses

The theory of principled knowledge and its role in performance and learning that constitutes the basis of the computer model that we describe in this report can be summarized as follows:

- Hypotheses about the nature of understanding:
  - Conceptual understanding of a procedure consists of knowledge about the task environment in which the procedure operates (rather than of the teleological semantics of the procedure).
  - Knowledge is declarative, i. e., goal-independent, context-free, and assertive (rather than procedural).
  - The type of declarative knowledge that is essential for procedural learning is knowledge of general principles (rather than knowledge of facts and episodes).
  - Principles constrain the possible states of affairs (rather than describe or predict).
- Hypotheses about performance:
  - A cognitive performance is a heuristic search through a problem space (rather than a problem reduction).
  - Procedural knowledge consists of collections of search heuristics (rather than of collections of subgoaling rules).
  - The function of principled knowledge in a heuristic search system is to facilitate the evaluation of search states (rather than to facilitate the generation of search states).
- Hypotheses about learning:
  - Learning is triggered when an incorrect or incomplete procedure generates a search state that violates one or more principles of the relevant domain (rather than, for instance, when two related rules fire in sequence).

- A faulty procedural rule is revised on the basis of information in the learner's principled knowledge (rather than on the basis of the information in a collection of instances).

As we have indicated in the presentation of these hypotheses, alternative hypotheses are possible with respect to each issue. In principle, each constellation of hypotheses define a possible cognitive model.[9] The particular choices we made in constructing the above theory were guided by our purpose of constructing a computational interpretation of the Conceptual Understanding Hypothesis. The next section describes a computer implementation of these hypotheses (*A Computer Model*, p. 19), and a later section describes the application of that model to the learning of arithmetic (*Computational Results*, p. 28).

[9]In practice, the design choices are not completely modular. A choice with respect to one issue sometimes limits the choices with respect to others. For instance, having choosen heuristic search as our performance theory, we are forced to assume that knowledge enters into either the generation or the evaluation of search states; there are no other options within that performance theory. The view of psychological theory construction as proceeding through successive decisions with respect to a set of *design issues* was first made explicit in Moore and Newell (1974), and has been developed further by Langley (1983a) and by VanLehn, Brown, & Greeno (1982).

A Computer Model

The theory presented in the previous section can be viewed as an abstract specification of an information processing system. A computer model of the theory is a runnable program that satisfies that specification. Implementation involves inventing computational mechanisms that work in accordance with the principles of the theory. We have implemented the *Heuristic Searcher* (HS), a computer model of the theory presented above. We first describe the performance system of the model and then its learning mechanism.

The performance system

HS is a production system architecture[10] augmented with a representation for principled knowledge. The system operates by searching a problem space. It selects an as yet unexpanded search state, and applies its current procedure to that state, thereby generating one or more new states. Search states are evaluated on the basis of their consistency with the system's principled knowledge.

Representation for procedural knowledge

A *procedure* in HS consists of a collection of production rules. The *condition* of a production rule is matched against the current search state. The *action* of a production rule consists of a single problem solving operator. An operator consists, in turn, of a *deletion list* and an *addition list*. When the operator is executed, the expressions in the deletion list are deleted from the current state and the expressions in the addition list are added, thereby creating a new search state.

Production rules encode search heuristics. The intended interpretation of rule R --> O is "if the current search state has property R, then consider operator O." There is no distinction in HS between search procedures and other kinds of procedures. An algorithm is a search procedure that is constrained enough to generate a single path through the problem space. Since the action side of the production rule consists of a problem solving operator, a production rule cannot write, edit, or delete expressions arbitrarily from working memory. Each computation performed has to correspond to a step through the problem space.

Representation for principled knowledge

Principles are represented in the HS system as *state constraints*. A state constraint C is an ordered pair $<C_r, C_s>$ of patterns, each pattern similar to the condition of a production rule. The left-hand pattern C_r is called the *relevance* pattern, because it determines the class of search states to which the constraint

[10]Production systems consist of collections of condition-action rules that are executed by (a) comparing their conditions with the contents of a working memory, (b) identifying those rules that have their conditions satisfied by the current contents of working memory, (c) selecting one or more of those rules, and (d) evoking the actions of the selected rule(s). Production systems were first used in cognitive psychology by Newell (1966) but are widely used in the analysis of human cognitive processes (Anderson, 1983; Newell and Simon, 1972; Klahr, Langley, & Neches, 1987; Laird, Rosenbloom, & Newell, 1986). Computer implemented production system languages were first proposed by Newell (1972, 1973). The reader who is unfamiliar with the production system format is referred to Neches, Langley, and Klahr (1987).

is relevant. The right-hand pattern C_s is called the *satisfaction* pattern, because it encodes the criterion that a state must match in order to satisfy the constraint (given that the relevance pattern matches). The relevance and satisfaction patterns are matched against the search states with the same pattern matcher that matches the production rule conditions. No new computational machinery has to be postulated in order to augment a production system architecture with this knowledge representation.

To illustrate the difference between the relevance pattern and the satisfaction pattern, consider the general principle *traffic should keep to the right side of the road.* This principle is violated if a person finds himself or herself on the left side of the road *while driving.* If the person is not driving, the principle is irrelevant. The HS system would encode this principle as *if HS is driving, then HS ought to be on the right side of the road.* If the current state does not contain the information that *HS is driving*, then the relevance pattern of the constraint does not match and the constraint is irrelevant. If the constraint is relevant, then two cases are possible. Either the current state contains the information *HS is on the right side of the road*, in which case the satisfaction pattern matches and the constraint is satisfied, or else the constraint is violated.

The operating cycle

The system takes one step forward in the problem space during each cycle of operation. A cycle begins by HS selecting an as yet unexpanded search state as the current state. The content of that state then becomes the effective working memory for that cycle. There is no other working memory than the selected search state. The system then matches all production rules in the current procedure against the selected state. One or more of those rules are evoked, and one or more new states generated. The system then matches its constraints against each new state, and records which constraints, if any, are violated by that state.

The reaction to a constraint violation depends upon whether the system is run in performance mode or in learning mode.[11] In *performance mode* HS executes a best-first search with the number of constraint violations as the cost function. The cost of a path is thus interpreted as the degree to which that path contradicts the system's principled knowledge, rather than as the amount of computational effort required to generate the path. Consequently, HS prefers solution paths that are more congruent with its principled knowledge over those that are less congruent.

In *learning mode* HS executes a breadth-first search, because it stops to learn as soon as it encounters a search state that violates a constraint. If a state violates some constraint, HS applies its learning mechanism to the rule that produced the constraint violation, thereby revising it. If there is more than one constraint violation, HS selects one of them at random to learn from. After revising a rule, HS backs up to the initial state and tries anew to solve the current problem.

[11]The HS system can also run in *diagnostic mode*. The details of how HS can be used in cognitive diagnosis will be reported elsewhere.

The HS learning mechanism

A mechanism for procedural learning performs some editing operation on a procedure in order to improve it. The HS learning mechanism operates by replacing single production rules with more constrained rules. Hence, it is a *form of discrimination learning* (Langley, 1983b, 1985, 1987). HS learns while doing, i. e., the learning mechanism operates in the context of the current state of the heuristic search. A mechanism for learning while doing must contain a specification of *when*--under what conditions--to pause and revise the procedure (the *triggering* problem), a criterion for *which* knowledge item to revise (the *assignment of blame* problem), and an algorithm for *how* to revise that item (the *revision* problem).

The triggering problem

Constraint violations indicate that the system's current procedure is not congruent with what the system knows about the task environment. Consequently, HS learns when it generates a search state that violates one or more state constraints. When a constraint violation occurs, the system terminates the current effort to solve its problem, applies its revision algorithm (see below), and then starts over from the initial state of the problem.

The assignment of blame problem

Given that the learning mechanism has been triggered, which rule should it revise? Which rule is to blame for the generation of the invalid state? The construction of the HS system implies that the constraint violation was produced by the rule that fired the operator that lead to the current state. This is shown by the following argument. Suppose that some operator further back in the search path generated an invalid state. That state would then have triggered the learning mechanism, HS would have revised the rule that lead to that state, and started over from the initial state. It would never have generated the current state. Hence, all states preceeding the current state are valid. The rule to revise is therefore the last rule to fire before the current state.[12]

The revision problem

Given a constraint violation the HS system tries to revise the rule that lead to the violation in such a way that future applications of that rule will not lead to violations of that constraint. The revision problem can be stated as follows:

[12]This argument presupposes that all errors are principled errors. The argument does not hold in domains where there are procedural errors. A procedural error is a step that does not violate any principle of the domain, but which nevertheless is not on the correct solution path. This point is discussed further in a later section (see p. 74).

> *Let S_1 and S_2 be two consecutive states in a search path. Hence, some production P with condition R was evoked in S_1, and fired some operator O with deletion list O_d and addition list O_a, thereby producing state S_2. Assume that S_2 violates constraint C with relevance pattern C_r and satisfaction pattern C_s, i. e., that S_2 matches the relevance pattern but not the satisfaction pattern. What is the appropriate revision of production P?*

Since HS learns as soon as it encounters a state that violates a constraint, S_1 does not violate C (or any other constraint). Hence, there are two types of constraint violations. In a *Type A* violation C is irrelevant in S_1, and it becomes relevant but not satisfied as the result of the application of operator O. In a *Type B* violation C is both relevant and satisfied in S_1, and remains relevant but becomes unsatisfied as the result of the application of O. We discuss the revision needed to handle the first type of constraint violation in detail.

Revision algorithm for a Type A violation. To repeat, in a Type A violation C is irrelevant in state S_1, and it becomes relevant but not satisfied in state S_2 as the result of the execution of operator O. If the relevance pattern C_r does not match S_1, but does match S_2, then the effect of executing operator O must have been to create expressions that enabled C_r to match. But since, *ex hypothesi*, the constraint C is violated in S_2, O did not create the expressions needed to complete the match for the satisfaction pattern C_s. This situation warrants two different revisions of the rule P that fired O. First, the condition of P should be revised so that the revised rule--call it P'--only fires in situations in which O will not complete the relevance pattern for C. Second, the condition of P should be revised so that the revised rule--call it P''--only fires in those situations in which *both* the relevance *and* the satisfaction patterns of C will become completed. The details of the two rule revisions are as follows:

- *Revision 1. Ensuring that the constraint is not relevant.* The purpose of the first revision is to avoid constraint violation by preventing constraint C from becoming relevant when operator O is executed. O will complete C_r when the parts of C_r that are not added by O are already present in S_1. Those parts are given by $(C_r - O_a)$, where the symbol "-" signifies set difference. To limit the execution of O to situations in which it will not complete C_r, we augment the condition of P with the negated expression

  $$not\ (C_r - O_a)$$

  In summary, if the expression $(C_r - O_a)$ matches the current state, then executing O will make C relevant, so we execute O only in situations in which that conjunction does *not* match.[13] The new rule created is:

  $$P': R \ \&\ not(C_r - O_a) \text{ --> } O$$

- *Revision 2. Ensuring that the constraint is satisfied.* The purpose of the second rule revision is to avoid constraint violation by forcing constraint C to become *both* relevant *and* satisfied

[13]The notation we use to describe the revision algorithm mixes set-theoretic notions like *set difference* with logical notions like *negation*. This should not cause any difficulties, because there is an obvious one-one mapping between sets of expressions and conjunctions of expressions: the set of expressions $(E_1, E_2, \ldots E_n)$ correspond to the conjunction $(E_1 \ \&\ E_2 \ \&\ \ldots \ \&\ E_n)$.

when O is executed. To guarantee that C_r will become complete, we augment the condition with the conjunction

$$(C_r - O_a).$$

To guarantee that C_s will also become complete we augment the condition of P with a conjunction that contains the parts of C_s that are not added by O. They are given by

$$(C_s - O_a).$$

Hence, the desired effect is achieved by appending the expression

$$(C_r - O_a) \cup (C_s - O_a)$$

to the condition of P, where the symbol "u" signifies set union. If this expression is present in the condition of a rule evoking O, then O is guaranteed to make the constraint C both relevant and satisfied. The new rule created is:

$$P'': R \cup (C_r - O_a) \cup (C_s - O_a) \rightarrow O$$

Summary of revision algorithm for Type A violations. If rule P with condition R evokes operator O in some state S_1 in which constraint C is irrelevant, thereby creating a new state S_2 in which constraint C is relevant but not satisfied, then we replace rule

$$P: \quad R \rightarrow O$$

with the two rules

$$P': \quad R \,\&\, \textit{not}(C_r - O_a) \rightarrow O$$

and

$$P'': \quad R \cup (C_r - O_a) \cup (C_s - O_a) \rightarrow O,$$

where "&" signifies conjunction, "-" signifies set difference and "u" signifies set union. The first rule limits the application of O to situations where C will not become relevant. The second rule evokes O in situations where C will become both relevant and satisfied. The section on computational results (*Computational Results*, p. 28) contains several detailed examples of how this learning algorithm functions in the context of learning arithmetic procedures.

The above description of the HS revision algorithm is simplified in the following respects: (a) We have not described the revisions needed to handle Type B violations, i. e., violations in which C is both relevant and satisfied in S_1, and becomes relevant but not satisfied in S_2 as the result of operation O. (b) In order to add parts of a constraint to a rule condition, correspondances must be established between the variables in the constraint and the variables in the rule. HS computes those correspondances by comparing the current *instantiation* of the rule to the current *instantiation* of the constraint.[14] (c) We have

[14]The way in which HS handles Type B violations and how it solves the problem of variable names are described in Ohlsson & Rees (1987).

not described the case in which the operator O deletes expressions. (d) Negated conditions can occur both in production rules and constraints. A negated condition can *cease* to match as the result of the addition of expressions to a search state, and the analysis has to be revised accordingly. (e) There are cases in which either of the two revisions results in the empty list of new conditions. In those cases only one new rule is created.

The definition of the learning algorithm is heavily influenced by the particular knowledge representation that we have chosen for the HS model. The key feature of the knowledge representation is the split between the relevance pattern and the satisfaction pattern. This feature implies the existence of two different revisions of the faulty rule, one that ensures that the constraint does not become relevant inappropriately, and one that ensures that the constraint is satisfied whenever it is relevant. If we had chosen a different representation for principled knowledge, we would have defined a different learning algorithm.

Discussion

HS share certain structural features with other production system architectures used to simulate cognitive processes.[15] Each system consists of an interpreter that matches a set of condition-action rules against a working memory, and selects one or more of the satisfied rules for execution. However, production systems differ with respect to the syntax of rules, the details of the matching process, the number of working memories, the method of conflict resolution, the learning mechanisms, etc. Four central features distinguish HS from other architectures: the simple mapping between architectural concepts and problem solving concepts, the separate representation for principled knowledge, the trade-off between generative and evaluative selectivity, and the rational learning mechanism. Each feature will be discussed in turn.

The first central feature of the HS system is that the architecture has been designed in accordance with the performance theory we are using. Constructs such as *operating cycle*, *production rule*, *working memory*, and *conflict resolution* belong to the theory of information processing systems (Newell & Simon, 1972, Chap. 2). A specification in terms of these constructs defines a particular, general-purpose information processing system. Although production system architectures are typically used to model human cognitive processes, they could, in principle, be used as general purpose programming languages. Constructs such as *search*, *heuristic*, *problem space*, *operator*, *search state*, and *evaluation function*, on the other hand, belong to the theory of problem solving (Newell & Simon, 1972, Chap. 2 and 3). A heuristic search system could, in principle, be implemented in any general purpose programming language such as Fortran or Lisp. There is no intrinsic connection between the concept of a production system and the concept of heuristic search.

---

[15]Production system architectures of the type to which HS belongs are sometimes called *neo-classical* in order to distinguish them from so-called *baroque* production systems used in expert systems research (Davis and King, 1976). The main difference between the two types of architectures is that in neo-classical systems the production rule is a procedural construct, while in baroque systems the production rule is a data-unit that is interpreted by unrestricted Lisp procedures. Neo-classical production systems languages are descendants from the PSG system developed by Newell (1972, 1973). They have recently been reviewed by Neches, Langley, & Klahr (1987). See also Langley (1983a) for an analysis of the space of production system architectures.

However, Newell (1980) has proposed the hypothesis that *problem solving is a fundamental category of human cognition*, i. e., that *all* central cognitive processes take the form of problem solving processes. Problem solving is *the* activity of the human cognitive architecture. This hypothesis implies that there should be a close relation between architectural constructs and problem solving constructs in models of human cognition. The mapping between architectural constructs and problem solving constructs is particularly straightforward in the HS system:

- Production rules correspond to search heuristics. The action of a production rule is constrained to be a single problem solving operator. Production rules cannot arbitrarily revise the contents of working memory. It is impossible to fire a production rule without taking a step through the problem space.
- The working memory is the current search state. The system has no working memory that is independent of the search process.
- Conflict resolution is done by state evaluation. All production rules that match the current state are evoked in parallell, thereby generating all possible descendants of the current state. A state is selected for expansion on the basis of its value on an evaluation function. There is no architectural process of conflict resolution that is independent of the problem solving process.
- An operation cycle consists of selecting a search state, matching the rules against that state, evoking all satisfied rules, and computing the evaluation function for each new state generated. In each operating cycle the system takes one step through the problem space. The system does not perform any other kind of computation.

In short, HS is an information processing architecture that has been designed in accordance with a particular theory of problem solving. The mapping between architectural constructs and problem solving constructs is similar in intent, but not identical in its details to the corresponding mapping in the Soar system (Laird, Rosenbloom, & Newell, 1986). The differences derive in part from our decision to represent principled knowledge as distinct from procedural knowledge.

The second central feature of the HS system is that principled knowledge is represented in the form of state constraints. A state constraint is a two-part pattern that a search state has to satisfy in order to be valid. The first part of the pattern is used to decide whether the constraint is relevant in a particular state or not; if so, then the second part of the pattern is used to decide whether the constraint is satisfied or not. State constraints have superficial similarities to several other computational constructs, but they function differently. A state constraint is not a production rule. It does not evoke motor actions, nor does it revise the content of working memory. A state constraint is not an inference rule; in particular, it is not a Horn clause.[16] The satisfaction pattern is not inferred or created when the relevance pattern matches. A state constraint does not guarantee that its right hand side *is* true when its left-hand side is true; it claims that

[16]A Horn clause is a restricted implicational formula in first-order predicate logic. The Horn clause is the representational format used in logic programming (Clark & Taernlund, 1982).

the right hand side *ought* to be true. State constraints have three different functions in the HS system: they constrain search during performance, they control when learning is to occur, and they serve as a source of information about how the current procedure should be revised. The notion of a state constraint is, as far as we know, unique to the work reported here. Other mechanisms for interfacing declarative and procedural knowledge have been proposed in the context of other simulation models. They will be discussed in later section (*Relations to Previous Research*, p. 60).

The third central feature of the HS system is that performance is guided by both generative and evaluative selectivity. *Generative selectivity* operates through strategic rules that propose good moves. Strategic rules improve the efficiency of search by focusing attention on the most promising *actions* in each state. *Evaluative selectivity* operates through evaluation functions that measure the promise of a state. Evaluation functions improve the efficiency of search by focusing attention on the most promising *states*. Confusingly, both strategic rules and evaluation functions are called *heuristics* in the search literature (Groner, Groner, & Bischof, 1983; Pearl, 1984). A. I. systems typically employ one or the other type of selectivity, but not both. The HS system operates with both generative and evaluative selectivity. Generative selectivity resides in the procedural knowledge (the production rules), while evaluative selectivity resides in the principled knowledge (the state constraints). The production rules generate actions, and the state constraints evaluate the states produced by those actions. The performance of the system is a function of both, and one type of selectivity can be traded off for the other. If either the procedure or the principled knowledge is correct and complete, correct performance will result. If both are deficient, performance may or may not be correct; the outcome depends on particular interactions between them.

The fourth central feature of the HS system is the rational learning mechanism. HS does not learn by being told the procedure it is trying to learn, nor by inducing it from a set of solved examples, nor by generalizing over a set of successful steps found by trial and error. The HS system constructs a procedure by constraining it to be consistent with the relevant principles. The state constraints control when learning is to occur: HS learns when a production rule generates a search state that violates some state constraint. By monitoring its performance with the state constraints, HS can know that a particular rule is faulty without being told by an outside source, and before it has completed even a single solution path. The revision of the faulty rule is guided by the particular way in which the relevant state violates the constraint. The required revision of the rule is derived from the constraint violation through the HS revision algorithm (see page 21). Principled knowledge enables HS to deduce the proper revision of the rule. This type of learning mechanism bears a family resemblance to other types of knowledge-based mechanisms, particularly to A. I. mechanisms for explanation-based learning (DeJong & Mooney, 1986; Mitchell, Keller, & Kedar-Cabelli, 1986) but it contrasts with the experience-oriented character of most mechanisms proposed in psychological theories of procedure acquisition.

The HS architecture is a model of the theory presented in the previous section in the sense that each hypothesis stated there is true of HS. However, HS is not the only possible model of that theory. In order to bridge the gap between the abstract hypotheses of the theory and the concrete details of the model,

auxiliary assumptions had to be introduced as implementation proceeded. The most global auxiliary assumption is that the human cognitive architecture is a production system. Although the production system format has extensive support from other modelling efforts, a model of our theory could have been implemented within some other type of information processing architecture. Even given the production system format, many details of the model could have been implemented differently. For instance, we choose to represent state constraints as binary patterns, and to relate them to search states through pattern matching. Clearly, there are other implementations of the idea that principles enable error detection. There are no hard and fast rules for how to construct *the* model for a particular theory.[17] The particular implementation reported here was choosen on a variety of criteria such as interest and simplicity. The justification for the implementation does not reside in the basis for the design decisions, but in the behavior of the resulting model.

A large number of hypotheses are required to specify an information processing architecture. It is almost impossible to derive predictions about the behavior of such a system by hand. The main purpose of fleshing out the hypotheses of the theory with the auxiliary assumptions required for implementation is precisely to use the implemented model to derive the behavioral predictions by running the model. The next section describes a sample of behaviors of the HS system in the context of the acquisition of arithmetic procedures.

[17]The difficulties associated with this aspect of computer simulation models have been discussed by Neches (1982), by Ohlsson (1988a), by VanLehn, Brown, & Greeno (1982), as well as by others.

Computational Results

The purpose of this section is to report three applications of the HS model that are relevant to the Conceptual Understanding Hypothesis. The first two applications demonstrate that HS can replicate the basic phenomena of children's learning in the domain of counting. First, we demonstrate that HS can construct a general counting procedure on the basis of the principles of counting, without receiving instruction in the procedure and without being given any solved examples. Second, we demonstrate that once HS has acquired a procedure for counting, the system can adapt that procedure to changes in the definition of the counting task. The third application demonstrates that the same mechanism that learns successfully in the domain of counting also learns successfully in the domain of symbolic algorithms: We verify that HS can cure itself from errors in its procedure for multi-column subtraction, if it is supplied with a state constraint representation of the conceptual basis for that procedure.

Constructing a procedure for an unfamiliar task

The basic claim of the Conceptual Understanding Hypothesis is that if a learner has principled knowledge about the environment in which a particular task appears, then he/she can discover a correct and general procedure for that task. The strongest evidence for this claim comes from the domain of counting. Our first application of HS shows that HS can construct a procedure for counting on the basis of a computational interpretation of the principles of counting. We describe the initial procedural knowledge of HS in this application, the principled knowledge, the learning process, and the outcome of the learning process.

Initial procedural knowledge for standard counting

To count a set of unordered objects is to repeatedly select an object from that set, increment the current number, and associate the new number with the selected object. When all objects in the set have been associated with numbers, the last number to be associated with an object is asserted to be the answer to the counting problem. Riley, Greeno, and Gelman (1984) call this task *standard counting*. Figure 1 shows a representational language for standard counting. The representation includes symbols for objects, sets, and numbers, and for a handful of properties and relations that are relevant for the counting task. Figure 2 shows a problem space for standard counting that builds on that representation. The problem space includes six operators, corresponding to the capabilities to select an arbitrary object from a set, to move attention from one object to another, to initialize counting at some number, to move attention from one number to another, to associate a number with an object, and to assert that a particular number is the answer to the current task. This set of capabilities is minimal in the sense that there is no smaller set that enables the learner to count; if one of these capabilities is missing, the learner is not ready to learn how to count. The initial state is encoded in the language defined in Figure 1. It contains a segment of the number line and some objects, some of which are members of the set of objects to be counted. The goal state is reached when some number as been identified as the answer to the counting problem.

Figure 3 shows an initial HS rule set for standard counting, as well as natural language paraphrases

Types of entities:
The representational language used in the counting application of HS assumes three types of entities:

- *objects* x_1, x_2 ...
- *numbers* n_1, n_2 ...
- *sets*

The HS model for standard counting considers a single set, namely the set of to-be-counted objects, called ToCountSet.

Properties:
There are four properties that apply to these entities:

- *First*
- *Current*
- *Answer*
- *Origin*

Both objects and numbers can have the properties of being the *first* object or number, and of being the *current* object or number (in a sequence of events). A sequence of events can only have one entity that has the property of being the *first* entity considered, and only one entity can be the *current* entity at any one point in time. Only a number can have the property of being an *answer*. The property of being the *origin* belongs to the smallest whole number the person knows. We assume in this application that it belongs to unity.

Relations:
There are four binary relations that hold between these entities:

- *Next*
- *Associate*
- *Member*
- *After*

Numbers are linked through the *next* relation. The expression (Next n_1 n_2) means that n_2 is the successor of n_1 in the number line. A number and an object can be *associated* with each other. An object can be a *member* of a set. In this application we only consider members of the set of to-be-counted objects. One entity can be considered *after* another entity (in a temporal sequence of events).

Figure 1: A representational language for standard counting.

of the rules. The initial rules impose minimal guidance on the application of the operators. Their main effect is to retrieve bindings for the operator arguments from working memory. Since the HS architecture is a search system, the collection of rules in Figure 3, although seriously incomplete, nevertheless constitutes an executable procedure. Execution of this procedure will generate ineffective but task relevant behavior. For instance, counting will be initialized at an arbitrarily chosen point in the number line (rule 3), and the number line will be traversed in random order (rule 4). Figures 2 and 3 together

The initial knowledge state:

The initial knowledge state for standard counting contains the number sequence (the numbers 1 through *n*, where 1 is marked as the *origin* and each number is linked to its successor with the *next* relation), the set ToCountSet of objects to be counted, and some additional objects that are not members of the ToCountSet. There is neither a current object nor a current number in the initial state.

Operators:

PickFirst(X) Declares object *x* as the *first* object; it thereby also becomes the *current* object.
The addition list O_a is {**(First X)(Current X)**}.
The deletion list O_d is empty.

PickNext(X_1, X_2) Moves the property of being the current object from x_1 to x_2. Also records the information that x_2 was attended to *after* x_1.
The addition list O_a is {**(Current X_2)(After X_2 X_1)**}.
The deletion list O_d is {**(Current X_1)**}.

Initialize(N) Declares the number *n* the *first* number; it thereby also becomes the *current* number.
The addition list O_a is {**(First N)(Current N)**}.
The deletion list O_d is empty.

Increment(N_1, N_2) Moves the property of being the current number from n_1 to n_2. It also records the fact that N_2 was considered *after* N_1.
The addition list O_a is {**(Current N_2)(After N_2 N_1)**}.
The deletion list O_d is {**(Current N_1)**}.

Associate(X, N) Associates the number *n* with the object *x*.
The addition list O_a is {**(Associate X N)**}.
The deletion list O_d is empty.

Assert(N) Asserts that the number *n* is the answer.
The addition list O_a is {**(Answer N)**}.
The deletion list O_d is empty.

Goal state:

The goal is to reach a state in which some number has the property of being the answer.

Figure 2: A problem space for standard counting.

constitute the initial procedural knowledge of HS in this application.

Principled knowledge for standard counting

Principled knowledge is encoded in HS in the form of state constraints, each constraint consisting of a relevance pattern and a satisfaction pattern. The state constraints for standard counting are shown in Figure 4 (Part 1 and Part 2). For each constraint the relevance pattern C_r is show to the left and the satisfaction pattern C_s tc the right, separated by the arbitrarily chosen symbol **. For simplicity, type designations like (Object X) and (Number N) have been left out of the statement of constraints. The

1. If *x* is any object, then select *x* as the first object.

   (Object X) ===> PickFirst(X)

2. If x_1 is the current object, and x_2 is any other object, then make x_2 the current object.

   (Object X_1)(Current X_1)(Object X_2) ===> PickNext(X_1, X_2)

3. If *n* is any number, then initialize counting at *n*.

   (Number N) ===> Initialize(N)

4. If n_1 is the current number, and n_2 is any other number, then switch to n_2 as the current number.

   (Number N_1)(Current N_1)(Number N_2) ===> Increment(N_1, N_2)

5. If *n* is the current number, and *x* is the current object, then associate *n* with *x*.

   (Number N)(Current N)(Object X)(Current X) ===> Associate(X, N)

6. If *n* is the current number, then assert that *n* is the answer.

   (Number N)(Current N) ===> Assert(N)

Figure 3: Initial rules for standard counting.

constraints are intended to capture the same ideas as the counting principles proposed by Gelman and Gallistel (1978), but our analysis differ from theirs in its details. We have broken down the counting principles into their component ideas and we have added some ideas that are not discussed by Gelman and Gallistel (1978).

The *One-One Mapping Principle* states that counting consists of establishing a one-to-one mapping between numbers and objects. As Part 1 of Figure 4 shows, we break this principle down into four component ideas: that an object is associated with at least one number, that an object is associated with at most one number, that a number is associated with at most one object, and that a number is associated with with at least one object. The *Cardinal Principle* states that the answer to a counting problem is the last number to be associated with an object. We break this principle down into three component ideas: that the size of a set cannot be known until all objects in the set have been associated with numbers, that the answer is a number associated with some object, and that the answer is the last number considered (see Part 1 of Figure 4). Our conception of the one-one mapping and cardinal principles is essentially the same as that of Gelman and Gallistel (1978). The difference is mainly that we are using a more fine-grained analysis of the ideas involved.

The *Stable Order Principle*, on the other hand, does not appear in our analysis. This principle says that the numbers used in counting must have a stable, repeatable order. We want to suggest that this

principle contains four distinct ideas. The first idea is that *the numbers form a linear ordering.* This idea is represented in the HS model (as in axiomatic theories of the number system) by the fact that the symbols for the numbers are linked together with the successor relation (called *Next*). This representation amounts to an assumption that children have a cognitive representation of the number line, an assumption that is supported by the available evidence (Resnick, 1983). Because the *Next* relations are stored in the model's memory, no state constraints are needed to encode this idea.

The second idea hiding in the Stable Order Principle is that the number line is traversed in a particular way during counting. For correct counting *the numbers must be generated in numerical order.* Once the number line has been stored in memory, it can be traversed in many different ways. For instance, it can be traversed by skipping every other number, by generating numbers in descending order, etc.. Also, traversal of the number line can, in principle, begin at any point along the line (although human beings may find some potential starting points easier to access than others). But the only way of traversing the number line that gives correct results in counting is to begin at unity and then follow the successor relations. We call this the *Regular Traversal Principle.* The state constraint representation breaks this idea down into four component ideas: Counting begins with the origin of the number line, each number considered is the successor of the previous number, the numbers are considered one at a time, and each number is associated with some object. The four state constraints corresponding to these ideas are shown in Part 2 of Figure 4.

The third idea hiding in the Stable Order Principle is that counting *imposes a linear ordering on the objects counted.* By assigning numbers, which have an intrinsic linear ordering, to objects, which do not, we are imposing a linear ordering on those objects. We call this idea the *Order Imposition Principle.* It is broken down into six component ideas: Only one object is designated as the first object in the ordering, objects are considered one at a time, no object is considered twice, an object is not considered after itself, the first object is never considered again,[18] and, finally, no object that is not a member of the to-be-counted set is considered. The six state constraints that encode these ideas are shown in Part 2 of Figure 4.

Finally, the actions of traversing the number line in the right way and imposing an order on the objects are not sufficient to produce correct counting. In addition, the two processes must be connected with each other in the right way. The fourth idea hiding in the Stable Order Principle is that *objects and numbers are associated with each other in the order in which they are attended to.* We call this the *Coordination Principle.* The state constraint representation for this idea is shown in Part 2 of Figure 4.

The state constraints in Figure 4 (Part 1 and Part 2) represent the principled knowledge of the HS system in this application. The set of constraints is not *unique.* Alternative formulations of the constraints are possible. Also, the set is not *minimal.* The constraints overlap in meaning. For instance, constraints

[18]The two constraints that an object is not to be considered after itself, and that the first object should never be considered a second time are, of course, special cases of the general constraint that no objects should be considered a second time.

A. The One-One Mapping Principle

1. An object should be associated with at most one number.

   (Associate X_1 N_1)(Associate X_1 N_2) ** (Equal N_1 N_2)

2. Every object considered during counting should be associated with some number.

   (Current X_1)(After X_1 X_2) ** (Associate X_2 N)

3. A number should be associated with at most one object.

   (Associate X_1 N_1)(Associate X_2 N_2) ** (Equal X_1 X_2)

4. For every number retrieved during counting there should be some object with which it can be associated.

   (Current N)(Not (Associate X_1 N)) ** (Current X_2)

B. The Cardinal Principle

1. A number is the answer to a counting problem only if there are no objects which are members of the to-be-counted set but which has not been associated with some number.

   (Answer N) ** (Not (Member X ToCountSet)(Not (Associate X N)))

2. The answer to a counting problem is one of the numbers associated with some object.

   (Answer N) ** (Associate X N)

3. The answer to the counting problem is the last number to be considered in the counting process.

   (Answer N) ** (Current N)

Figure 4: State constraints for standard counting, Part 1.

B1 (see Figure 4, Part 1) and C4 (see Figure 4, Part 2) express the idea that *all objects should be counted* in two different ways. Also constraints D4 and D5 are special cases of D3. Overlap in the meaning of state constraints implies that learning from one constraint may make learning from another constraint unnecessary. As a result, all constraints are not involved in every learning run. The set of state constraints in Figure 4 is *complete*. It is sufficient to determine correct counting.

The learning process

HS takes different paths through the procedure space on different learning runs, for two reasons. First, if HS generates more than one state that violates some constraint on a particular cycle, it selects one at random to learn from. Second, since the domain theory is not minimal, learning from one constraint may preempt learning from another constraint. Hence, the order in which constraint violations are noted by the system influences the path through the procedure space. The final procedures learned in different learning runs are, of course, very similar, but not identical.

C. The Regular Traversal Principle

1. Initialize counting at the first number in the number line.

   (First N) ** (Origin N)

2. Consider one number at a time.

   (Current N_1)(Current N_2) ** (Equal N_1 N_2)

3. The numbers should be considered in the order defined by the *next* relations.

   (Current N_1)(After N_1 N_2)(Not (Equal N_1 N_2)) ** (Next N_2 N_1)

4. For each number considered, the preceeding number should be associated with some object (i. e., use all numbers).

   (Current N_1)(Next N_2 N_1) ** (Associate X N_2)

D. The Order Imposition Principle

1. Initialize counting with a single object.

   (First X_1)(First X_2) ** (X_1 = X_2)

2. Do not consider an object that is already associated with a number.

   (Current X)(Not (Current N)) ** (Not (Associate X N))

3. Do not cycle back to the first object.

   (First X_1) ** (Not (After X_1 X_2))

4. Do not consider an object after itself.

   (After X_1 X_2) ** (Not (Equal X_1 X_2))}

5. Consider only one object at a time.

   (Current X_1 X_2)) ** (Equal X_1 X_2)

6. Do not consider objects that are not in the set of to-be-counted objects.

   (Current X) ** (Member X ToCountSet)

E. The Coordination Principle

1. Numbers and objects are associated with each other in the order in which they are considered.

   (Current X)(Current N_1)(Associate X N_2) ** (Equal N_1 N_2)

Figure 4: State constraints for standard counting, Part 2.

We will analyze a particular learning experiment in which HS was started with the initial rule set for counting shown in Figure 3 and the state constraints shown in Figure 4 and was given practice on counting a set with three objects. During learning the model commits the types of counting errors observed in childrens' performance, such as counting an object more than once, skipping numbers, and choosing the wrong number as the answer. It successively corrects these errors by noticing violations of the state constraints, and revising the initial rules accordingly.

As an example of the construction of a rule, consider rule 6 (see Figure 3): *If n is the current number, then assert that n is the answer.* This rule will prematurely assert that the current number is the answer when there are still objects left to be counted. HS learns the correct rule by transforming rule 6 in two steps. Figure 5 shows a graph representation of the path through the rule space for this particular rule. Learning proceeds from top to bottom. At the top of the figure is the formal version of the initial rule as stated in Figure 3. The vertical arrows represent learning steps. At the head of the arrow is the condition or conditions that were added to the rule in that step. Each learning step is triggered by the violation of a state constraint. The constraint is shown to the right of the vertical arrows. The labels on the constraints refer to Figure 4. The final rule is shown at the bottom of the graph. The reader who intends to follow the description how of the correct rule is learned in detail may want to review the HS learning algorithm (p. 21) at this point.

The first learning step is triggered when the initial rule violates constraint B2: *The answer to a counting problem is one of the numbers associated with some object.* The formal version of this constraint is shown to the right in Figure 5. Suppose that, say, 2 is the current number. The condition side R of rule 6 then becomes instantiated to:

$$R = \{(\text{Number } 2)(\text{Current } 2)\}$$

The addition list O_a of operator **Assert** (see Figure 2, p. 30) is then equal to

$$O_a = \{(\text{Answer } 2)\}$$

while the deletion list O_d is empty. The constraint is irrelevant before the **Assert** operator is fired, so we have a Type A constraint violation, in which the execution of the an operator makes the constraint relevant but not satisfied. Two revisions of the faulty rule are attempted.

Revision 1. Ensuring that the constraint is not relevant. The HS learning algorithm first tries to construct the expression

$$not(C_r - O_a).$$

However, in this case C_r is equal to

$$C_r = \{(\text{Answer } 2)\}$$

so the relevance pattern and the addition list are identical. Hence, the expression

$$not(C_r - O_a) = \{(\text{Answer } 2)\} - \{(\text{Answer } 2)\}$$

which is equal to the empty set, so no new rule can be created in this revision.

Revision 2. Ensuring that the constraint is satisfied. Next, the learning mechanism tries to construct the expression

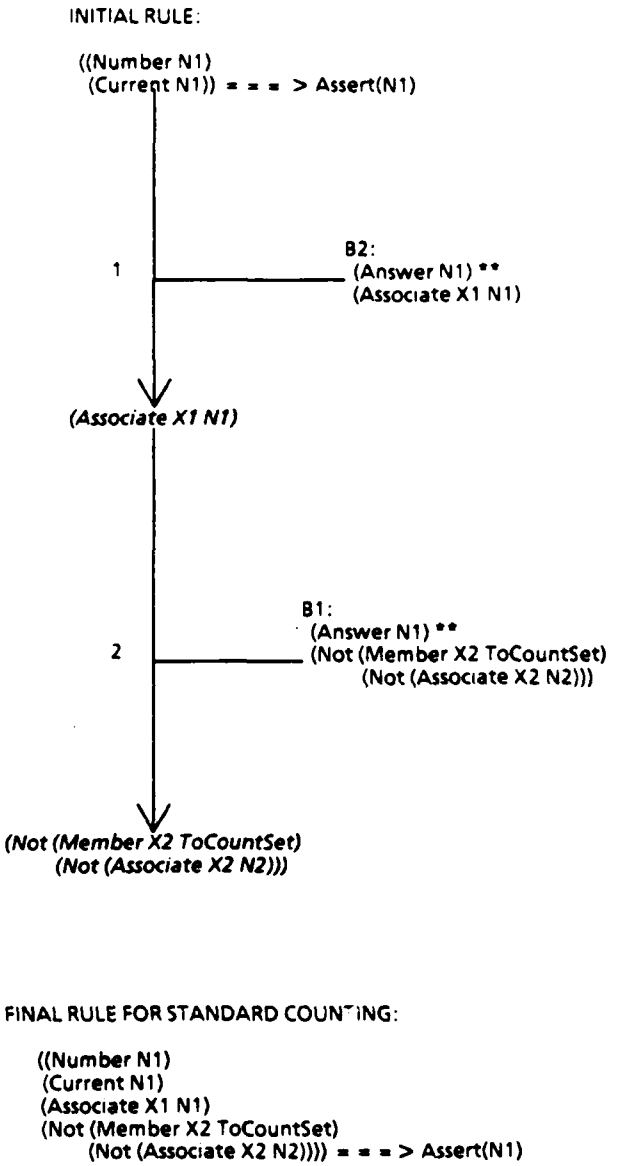


Figure 5: A learning path for rule 6 (see Figure 3).

$$(C_r - O_a) \cup (C_s - O_a).$$

The left-hand term is, as we just showed, empty, so this expression reduces to

$$(C_s - O_a).$$

The satisfaction pattern C_s is in this case equal to

$$C_s = \{(\text{Associate } X_1\ 2)\}$$

where X is some object. This expression will not change by the subtraction of $O_a = \{(\text{Answer } 2)\}$, so we have

$$(C_s - O_a) = \{(\text{Associate } X_1\ 2)\}$$

Proper substitution of variables for constants leads to the expression (Associate X_1 N_1), which is added to the rule.[19] In other words, the learning mechanism adds the condition that *the number designated as the answer has to be assigned to some object* to the rule. The formal version of this condition is shown on the path in Figure 5, at the head of the vertical arrow.. Having revised the rule HS backs up to the initial state, and tries to do the counting task again, using the new rule instead of the original rule.

In the second learning step, the revised rule violates constraint B1 (see Figure 4): *A number is the answer to a counting problem only if there are no objects which are members of the to-be-counted set but which has not been associated with some number.* The rule is now constrained to select only numbers that have been assigned to objects, but it does not yet know that it has to wait until all objects have been counted. It prematurely asserts that the current number is the answer, as soon as that number has been assigned to an object. This is a Type A violation, because the constraint is not relevant until the operator **Assert** has been fired. As in the previous learning step, the expression

$$not(C_r - O_a)$$

is empty so Revision 1 does not lead to the creation of a new rule. In Revision 2 HS constructs the expression

$$(C_r - O_a) \cup (C_s - O_a)$$

which is equal to

$$(C_s - O_a).$$

Since

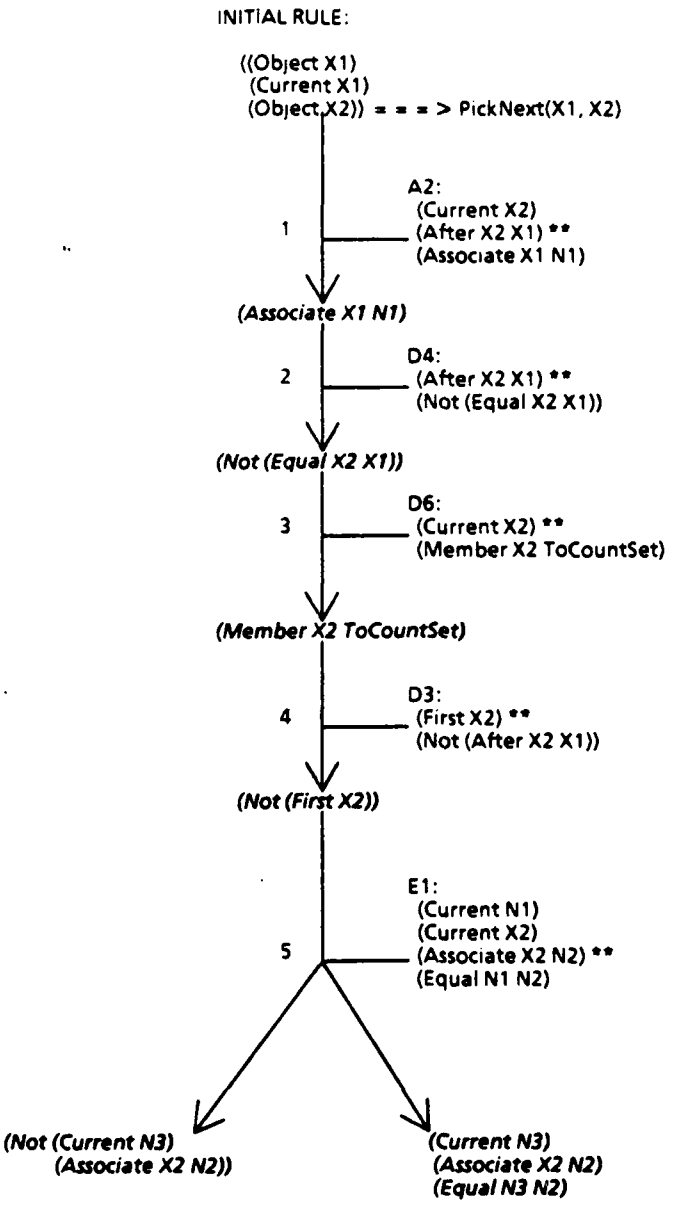
$$C_s = \{(\text{Not (Member X ToCountSet)(Not (Associate X N))})\}$$

and

$$O_a = \{(\text{Answer N})\}$$

the subtraction of the addition list from the satisfaction pattern simply gives the satisfaction pattern unchanged. Therefore, the expression added to the rule in Revision 2 is equal to C_s. In short, the learning mechanism adds the condition element *the set of to-be-counted objects is empty*, or, formally, *it should not be the case that there exists an object which is a member of the to-be-counted set and which has not been assigned a number.* The condition of the rule then becomes as shown in the bottom of

[19] In order for the new expression to interface correctly with the previous expressions in the rule, HS has to coordinate the variable names. The computations involved in the coordination of variable names are not described in this report, but see Ohlsson & Rees (1987). In this report we will simply assume that the variables are given the correct names.



FINAL RULE FOR STANDARD COUNTING:

((Object X1)
(Current X1)
(Object X2)
(Associate X1 N1)
(Not (Equal X2 X1))
(Member X2 ToCountSet)
(Not (First X2))
(Not (Current N3)
(Associate X2 N2))) = = = > PickNext(X1, X2)

Figure 6: A learning path for rule 2 (see Figure 3).

Figure 5. The result of this second learning step is a correct rule.[20]

The learning of the correct rule for asserting the answer is a particularly simple example of a rule transformation. The initial rule only has to be extended with two additional conditions, and only one new rule is created in each learning step. The rule for selecting the next object, rule 2 in Figure 3, presents a more complex case. Figure 6 shows the construction of the correct version of this rule. As in Figure 5, the initial rule is shown at the top of the figure, the constraints that are violated are shown to the right of the path, and the conditions added to the rule are shown along the path. The final, correct, rule is shown at the bottom of the figure. Five learning steps are required to construct the correct rule in this particular simulation run.

In the first learning step rule 2 violates the constraint that each object has to be associated with at least one number (see Part 1 of Figure 4, constraint A2). This happens because the system moves attention from one object to the next without counting it. This constraint violation follows the same pattern as the ones analyzed previously. It is a Type A violation, where the first revision does not yield a new rule, and the second revision consists of adding the satisfaction pattern of the constraint to the rule. Since C_s in this case is

$$C_s = \{(\text{Associate X N})\}$$

the learning mechanism adds the constraint that the current object has to be counted before a new current object can be selected. The next two violations follow the same pattern. The revised rule violates the constraint that objects should not be counted repeatedly, and so the learning mechanism adds the condition that counted objects should not be selected for counting again:

$$C_s = \{(\text{Not (Equal } X_2\ X_1))\}$$

The rule next selects some object that is not in the set of objects to be counted, and so is constrained to deal only with those objects:

$$C_s = \{(\text{Member X ToCountSet})\}$$

In the fourth learning step the revised rule violates the constraint that it should not return to the first object (see constraint D3 in Part 2 of Figure 4). This is yet another Type A violation, but in this case Revision 1 yields a new rule but Revision 2 does not. Since in this case

$$C_r = \{(\text{First } X_1)\}$$

and

$$O_a = \{(\text{After } X_1\ X_2)(\text{Current } X_1)\}$$

the expression

$$(C_r - O_a)$$

is instantiated to

$$\{(\text{First } X_1)\} - \{(\text{After } X_1\ X_2)(\text{Current } X_1)\}$$

[20]The third constraint that expresses the Cardinality Principle (constraint B3 in Part 1 of Figure 4) was not violated in this learning run. This illustrates the earlier comment that the overlap in meaning between state constraints implies that learning from one constraint may preempt learning from another.

so we have

$$not(C_r - O_a) = (\text{Not (First } X_1))$$

which is added to the rule. This condition prevents the rule from firing when the object it considers making the current object was, in fact, the first object counted. Revision 2 illustrates the complexities introduced by negation. The satisfaction pattern of the relevant constraint is a negated pattern, and it happens to be the case that the operator **PickNext** adds the positive part of that pattern to the state. Hence, Revision 2 cannot succeed. There is *no* way of revising the rule so that both the relevance pattern and the satisfaction patterns are guaranteed to be true. In fact, whenever the **Assert** operator fires, the satisfaction pattern is guaranteed to be false. The learning mechanism recognizes that the operator adds the negation of the satisfaction pattern, and does not create a second rule for this violation.

Finally, in the fifth learning step, the rule gets out of step, as it were, and violates constraint E1 (see Part 2 in Figure 4) which says that *numbers and objects are associated with each other order in which they are generated.* This is, once again, a Type A violation, but in this case *both* revisions generate non-empty extensions of the rule, so two new rules are created.

Revision 1. Ensuring that the constraint is not relevant. We have

$$C_r = \{(\text{Current } N_1)(\text{Current } X_1)(\text{Associated } X_1\ N_2)\}$$

and

$$O_a = \{(\text{After } X_1\ X_2)(\text{Current } X_1)\}$$

Hence, the expression

$$not(C_r - O_a)$$

is in this case equal to

$$not[\{(\text{Current } N_1)(\text{Current } X_1)(\text{Associated } X_1\ N_2)\} - \{(\text{After } X_1\ X_2)(\text{Current } X_1)\}]$$

which reduces to

$$not((\text{Current } N_1)(\text{Associated } X_1\ N_2)).$$

This expression is added to the rule. The final result is a rule that says "If the current object has been associated with a number, and there is a second object that is a member of the set of objects to be counted, but that is not the first object, and that has not been associated with a number, then move attention to that second object", which is the correct rule, shown at the bottom of Figure 6.

Revision 2. Ensuring that the constraint is satisfied. Next, we have the expression

$$(C_r - O_a) \cup (C_s - O_a)$$

which does not reduce to the empty list in this case. The part $(C_r - O_a)$ is, as we have seen above, equal to

$$\{(\text{Current } N_1)(\text{Associated } X_1\ N_2)\}.$$

The expression $(C_s - O_a)$ becomes

$$\{(\text{Equal } N_1\ N_2)\} - \{(\text{After } X_1\ X_2)(\text{Current } X_1)\}$$

which reduces to

$$\{(\text{Equal } N_1\ N_2)\}.$$

Hence, the set union of the two expressions is equal to

$$\{(\text{Current } N_1)(\text{Associated } X_1\ N_2)(\text{Equal } N_1\ N_2)\}$$

which is then added to to create a second new rule.

The rule created in Revision 2 of the fifth and last learning step is not a correct rule, but a so-called monster rule. It is a syntactically correct and executable rule which is simply not part of correct counting. The rule says that *if the current object x_1 has been assigned to the current number n, and some other object x_2 has previously been assigned to n, then select x_2 as the next object*, which is a manifestedly incorrect counting rule. The rule is harmless, i. e., it will never fire, if all the other rules are correct, because two objects will never be assigned to the same number. However, if other rules are also incorrect, then this rule might fire. It will generate the error of going back and counting a previously counted object as second time.

Although we have analyzed the construction of rules 6 and 2, respectively, as sequences of learning steps, those steps did not occur on successive trials during the learning run. HS does not first go through all required revisions of one rule, and then turn to another rule, etc. The learning steps required to construct the correct versions of rules 2 and 6 occurred interspersed among the learning steps required to construct the other rules. The order of learning steps is determined by the order in which HS encounters violations of constraints. In order to make the learning process easier to follow, Figures 5 and 6 abstract out the revisions of rules 6 and 2, respectively, from the trace of the simulation run, and presents them in sequence. This is an exposition technique, it is not how HS learns.

An overview of the entire learning process is given in Table 1. The particular learning run analyzed here required twenty-two trials. HS practiced on a set of three objects. Each trial consists of a problem solving attempt in which HS executes its procedure until a constraint violation is discovered, revises the faulty rule, and starts over. The twenty-two learning trials were accomplished in 97 production system cycles. Each line in Table 1 corresponds to one trial. The first column shows the trial number. The second column shows the number of cycles before a constraint violation was detected for each trial. As the table shows, the number of cycles increases over trials. HS gradually performs larger and larger portion of the task correctly. The third column shows the constraint that was violated in that trial. The violated constraint is the constraint that HS learned from in that trial. Finally, the last column represents the six rules with the digits 1 through 6; the rule that was revised on that trial corresponds to the bracketed number. In the twenty-third trial (not shown in the table), HS counted correctly the set of three objects. The correct solution to the problem of counting three objects required eleven production system cycles.

As the table shows, the two learning steps that transformed rule 6 into the correct rule occurred on trials 8 and 12, respectively, while the five learning steps required to learn the correct form of rule 2 are spread out over the entire learning process, beginning with trial 5 and ending with trial 22. The table also shows that a constraint can be violated by several different rules. For instance, constraint E1 is violated

Table 1: Overview of the learning process for standard counting.

Trial no.	No. of cycles before constraint violation	Constraint violated	Rules 1-6; revised [x]
1	1	A4	1 2 [3] 4 5 6
2	1	D6	[1] 2 3 4 5 6
3	2	D1	[1] 2 3 4 5 6
4	2	C4	1 2 [3] 4 5 6
5	2	A2	1 [2] 3 4 5 6
6	3	C4	1 2 3 [4] 5 6
7	3	D4	1 2 3 [4] 5 6
8	3	B2	1 2 3 4 5 [6]
9	4	D4	1 [2] 3 4 5 6
10	4	D6	1 [2] 3 4 5 6
11	4	E1	1 2 [3] 4 5 6
12	4	B1	1 2 3 4 5 [6]
13	4	E1	1 2 3 [4] 5 6
14	5	D1	1 2 [3] 4 5 6
15	5	D1	1 2 [3] 4 5 6
16	5	A3	1 2 3 4 [5] 6
17	6	D4	1 2 3 [4] 5 6
18	6	D3	1 2 3 [4] 5 6
19	7	D3	1 [2] 3 4 5 6
20	7	D3	1 2 [3] 4 5 6
21	9	C3	1 2 3 [4] 5 6
22	10	E1	1 [2] 3 4 5 6

by rules 2 (trial 22), 3 (trial 11), and 4 (trial 13). The table also shows that not all constraints are involved in the learning run. For instance, constraint D2 was not violated in this particular run. The particular learning process HS goes through on the way to mastery of standard counting is a function of the representation, the initial rules, the state constraints, and the order in which violations are discovered. Different simulation runs will yields slightly different learning processes.

The learning outcome

The final outcome of learning is a procedure for standard counting that counts correctly. It consists of six rules, corresponding to the six rules in the initial procedural knowledge (see Figure 3), but with the conditions revised in such a way as to produce correct performance. The final rules are shown in Figure 7 (Part 1 and Part 2). The level of generality of the learned counting procedure is the same as the level of generality of the constraints. The learned procedure transfers to arbitrarily large sets, i. e., to sets it has not practiced on.

The outcome of the above learning simulation is in accord with the empirical data from the counting domain, as well as with the Conceptual Understanding Hypothesis: HS is able to discover the correct procedure for standard counting without being given a description of the procedure, without seeing any solved examples, and without being given an explanation of the procedure. The procedure is constructed incrementally in an effort to avoid violating the counting principles. The Conceptual Understanding Hypothesis also claims that procedures constructed in this way are flexible when confronted with a variation of the relevant task. The next application deals with this phenomenon.

Adapting a procedure to a change in a familiar task

Life rarely presents us with totally new tasks. There are always some similarities between a new task and some previously mastered task. The Conceptual Understanding Hypothesis claims that if a procedure has been constructed on the basis of principled knowledge of the task environment, then the learner should be able to adapt that procedure to a conceptually equivalent but procedurally different task. Hence, in our second application we verify that the counting principles enables HS to adapt its procedure for standard counting to two changes in the task. First, we modify the standard counting task by requiring that the objects be counted in a predefined order (*ordered counting*). Second, we modify the standard counting task by requiring that the objects are counted in such a way that a particular object is assigned a particular number (*constrained counting*). Empirical research has shown that children can readily adapt to these two non-standard counting tasks (Gelman & Gallistel, 1978; Gelman & Meck, 1983, 1986; Gelman, Meck, & Merkin, 1986). In both simulations, we first have HS discover the procedure for standard counting in the way analyzed in the previous subsection. Then we change the task, and observe how HS transfers the old procedure to the new task.

Transferring from standard to ordered counting

In ordered counting, objects are counted in accordance with some predefined ordering. Ordered counting differs from standard counting with respect to the selection of objects. Rather than selecting any

1. If x_1 is any object, x_1 is a member of the ToCountSet, and no object has yet been selected as the first object, then select x_1 as the first object.

   (Object X_1)(Member X_1 ToCountSet)(Not (Object X_2)(First X_2))
   ===> PickFirst(X_1)

2. If x_1 is the current object, x_1 has been associated with some number n_1, x_2 is any other object, x_2 is a member of the ToCountSet, x_2 is not the first object, and it is neither true that there is a current number n_2, nor that x_2 has been associated with some number n_3, then switch to x_2 as the current object.

   (Object X_1)(Current X_1)(Associate X_1 N)(Object X_2)(Member X_2 ToCountSet)
   (Not (First X_2)) (Not (Current N_2)(Associate X_2 N_3)) ===> PickNext(X_1, X_2)

3. If n_1 is any number, there is no object x_1 such that n_1 been associated with x_1, some object x_2 has been selected as the curent object, and there is no number n_2 such that n_1 is the successor to n_2, then begin counting with n_1.

   (Number N_1)(Not (Object X_1)(Associate X_1 N_1))(Object X_2)(Current X_2)
   (Not (Number N_2)(Next N_2 N_1)) ===> Initialize(N_1)

4. If n_1 is the current number, n_2 is any other number, n_1 is the predecessor of n_2, n_3 is associated with some object x, x is not the current object, and n_3 is the predecessor of n_2, then switch to n_2 as the current number.[21]

   (Number N_1)(Current N_1)(Number N_2)(Next N_3 N_2)(Associate X N_2)(Not (Current X))
   (Not (Equal N_2 N_1))(Next N_1 N_2) ===> Increment(N_1, N_2)

5. If n is the current number, and x_1 is the current object, and n has not been associated with any other object x_2, then **Associate** n with x_1.

   (Number N)(Current N)(Object X_1)(Current X_1)(Not (Associate X_2 N))
   ===> Associate(X_1, N)

6. If n_1 is the current number, and n_1 has been associated with some object x_1, and there is no object x_2 in the ToCountSet that has not been associated with some number n_2, then assert that n_1 is the answer.

   (Number N_1)(Current N_1)(Object X_1)(Associate N_1 X_1)
   (Not (Object X_2)(Member X_2 ToCountSet)(Not (Number N_2)(Associate X_2 N_2)))
   ===> Assert(N_1)

Figure 7: Final rules discovered by HS for standard counting.

[21] The formulation of this rule is opaque, because it introduces two symbols, n_1 and n_3, for the same number. The two conditions that claim that n_1 and n_3 are predecessors to n_2 constitute an implicit equality-test that binds together the expressions in the rule condition. If the program knew the meaning of the predecessor relation, it could, in principle, transform the rule into a less opaque form. However, the rule as stated here is the form that was actually learned in the particular learning experiment we are reporting.

F. Ordering Constraints

1. Objects are considered from left to right.

   (Current X_1)(After X_1 X_2)(Adjacent X_1 X_3)(LeftOf X_3 X_1) ** (Equal X_1 X_3)

2. Objects are associated with numbers in order from left to right.

   (Current X_1)(Object X_1)(Object X_2(Adjacent X_2 X_1)(LeftOf X_2 X_1) ** (Associate X_2 N)

Figure 8: Constraints that define ordered counting.

object, the system now has to select one according to certain criteria. The ordered counting task was defined for HS by extending the inputs to the program in two ways. First, we extended the initial knowledge state by adding *left of* and *adjacent* relations between the objects, thereby imposing an order on the set of objects to be counted. Second, we extended the principled knowledge of the model. In unordered counting, the act of counting imposes a linear ordering on a set of objects that does not have an intrinsic order. In ordered counting, however, the set of to-be-counted objects has an ordering given to it by the setting, and the task is to traverse that order. In this application HS was required to count objects in order from left to right. Two new constraints express this idea. The first order constraint says that *if the current object is considered after another object, then it should be immediately to the left of that object*, i. e., objects should be considered in order from right to left. The second order constraint says that *objects should be assigned numbers according to the given order*, which in this case means from left to right. The state constraint representation of these two ideas is shown in Figure 8.

In this simulation experiment HS first learned the procedure for standard counting in the way described in the previous subsection. We then posed the task of counting the objects in order from left to right and run the system again on this new task. Some of the rules HS learned for standard counting task are correct for ordered counting as well. The rules for initializing counting at unity, for incrementing the counting number, for associating a number with an object, and for asserting a number as the answer are all correct for the ordered counting task. But the two rules for selecting a first object and for selecting a next object produce constraint violations, and are revised to fit the new task.

For instance, rule 2, the rule that selects the next object, has no conditions that constrain it to select objects in order from left to right. Figure 9 shows the search through the rule space for rule 2 in this application. The top part of the figure, before the box labelled "Adaptation to ordered counting", shows the initial construction of rule 2 and is identical with Figure 6. The learning step inside the box is caused by the rule violating ordering constraint F1 (see Figure 8).

Two new rules are created in this learning step. The rule created by Revision 2, shown at the bottom and to the right in Figure 9, is the correct rule for selecting the next object in the ordered counting task.

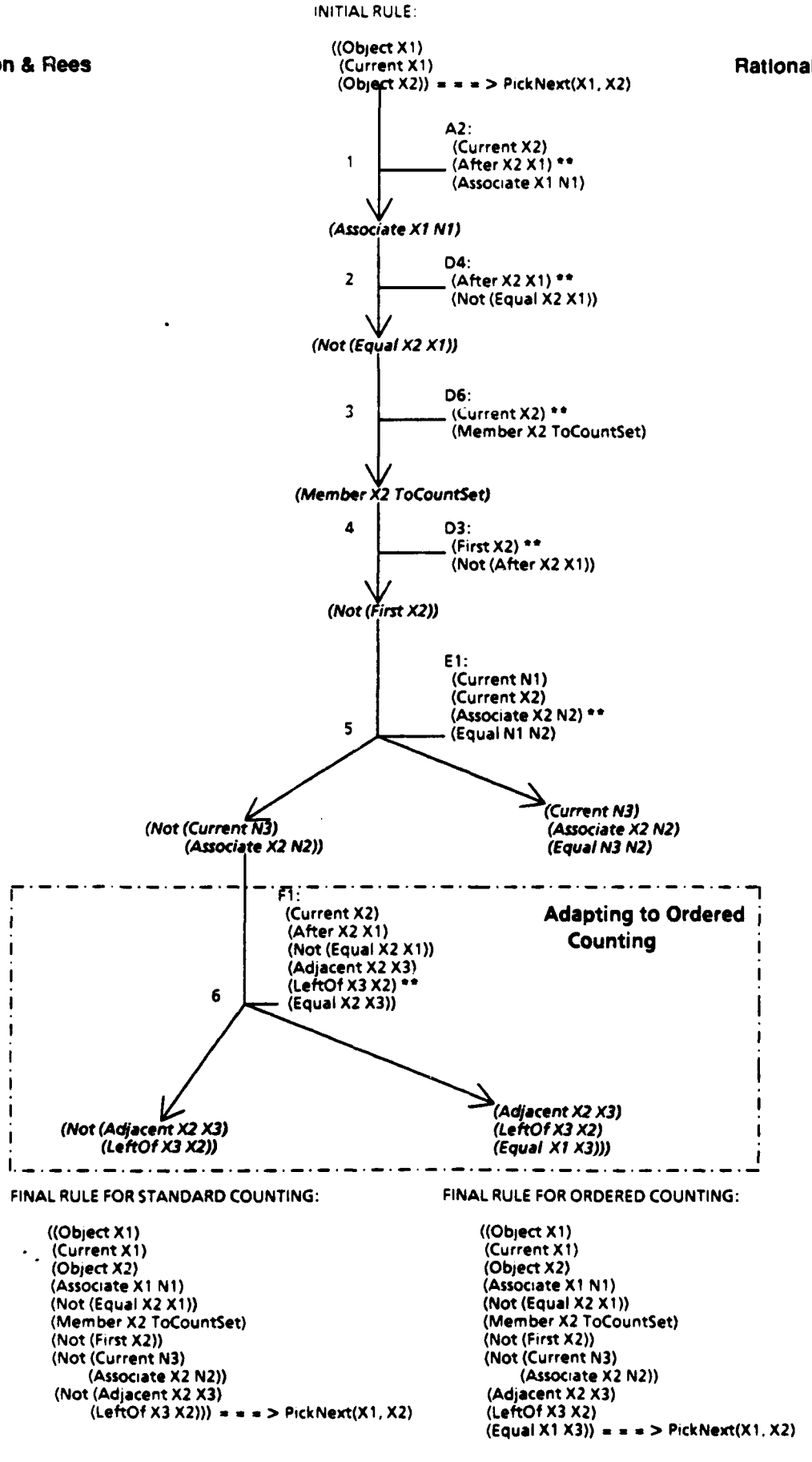


Figure 9: *Revisions of rule 2 (see Figure 7) in adaption to ordered counting.*

The rule created in Revision 1, shown at the bottom and to the left in Figure 9, is a modification of the rule for standard counting. The performance of this rule will depend on the perceptual encoding of the problem situation. If the initial knowledge state encodes the objects to be counted as unordered, this rule will function correctly. Hence, the outcome of this learning step is a procedure that can solve *both* task correctly. However, if the initial state contains information about the ordering relations of the objects to be counted, then this rule will refuse to fire. This amounts to a prediction that having adapted to ordered counting, the learner cannot perform unordered counting *if he/she pays attention to the ordering relations between the objects*. After this adaptation the system will always count according to the ordering relations between the objects, if those are encoded in the initial state.

Without principled knowledge about the task--without a representation of the task that is more abstract than the rules themselves--there is no way of knowing which rules are still relevant and which are not when the task is changed. Therefore, an empirical learning system would have to construct a new procedure from scratch for the new variant of the task. HS, on the other hand, knows that a rule needs to be revised only if it produces constraint violations, but not otherwise. Hence, it can back up the minimal distance in the procedure space that is needed to transfer its current procedure to the new task. The construction of the procedure for standard counting required twenty-two learning steps, but the adaptation to the ordered counting task only requires two learning steps. HS shows considerable transfer from one task to the other.

The ability of HS to adapt to a change in the task does not depend on the particular characteristics of the switch from unordered to ordered counting. For instance, it does not depend on the fact that this switch involves the *addition* of constraints. In a different learning experiment HS learned to adapt in the opposite direction. In this experiment HS began by constructing the procedure for ordered counting. We then switched the task to standard counting. Figure 10 shows the path through the rule space for rule 2 in this learning experiment. The initial construction of the correct rule for ordered counting is shown along the right branch of the figure. It consists of three learning steps, caused by the violation of constraints D3, F1, and A2, in that sequence. The final, correct, rule for ordered counting is shown to the right in the figure.

As the figure shows, learning step 2 produces a pair of rules, only one of which is the correct rule for ordered counting. When HS is confronted with the standard counting task, the system backs up in the rule space to this point, and fires the other rule produced in learning step 2. This rule, a supposedly "incorrect" rule generated during the learning of ordered counting, is developed into the correct rule for unordered counting in three further learning steps, shown inside the box labelled "Adaptation to unordered counting" in Figure 10. Hence, the final result is again a procedure that can do both standard counting and ordered counting correctly.

The third learning step inside the box (labelled step 6 in the figure) produces two rules, one of which is the final rule for standard counting. The other rule is yet another example of a rule created during learning that is not part of the correct procedure. It does not fire in either standard or ordered counting,

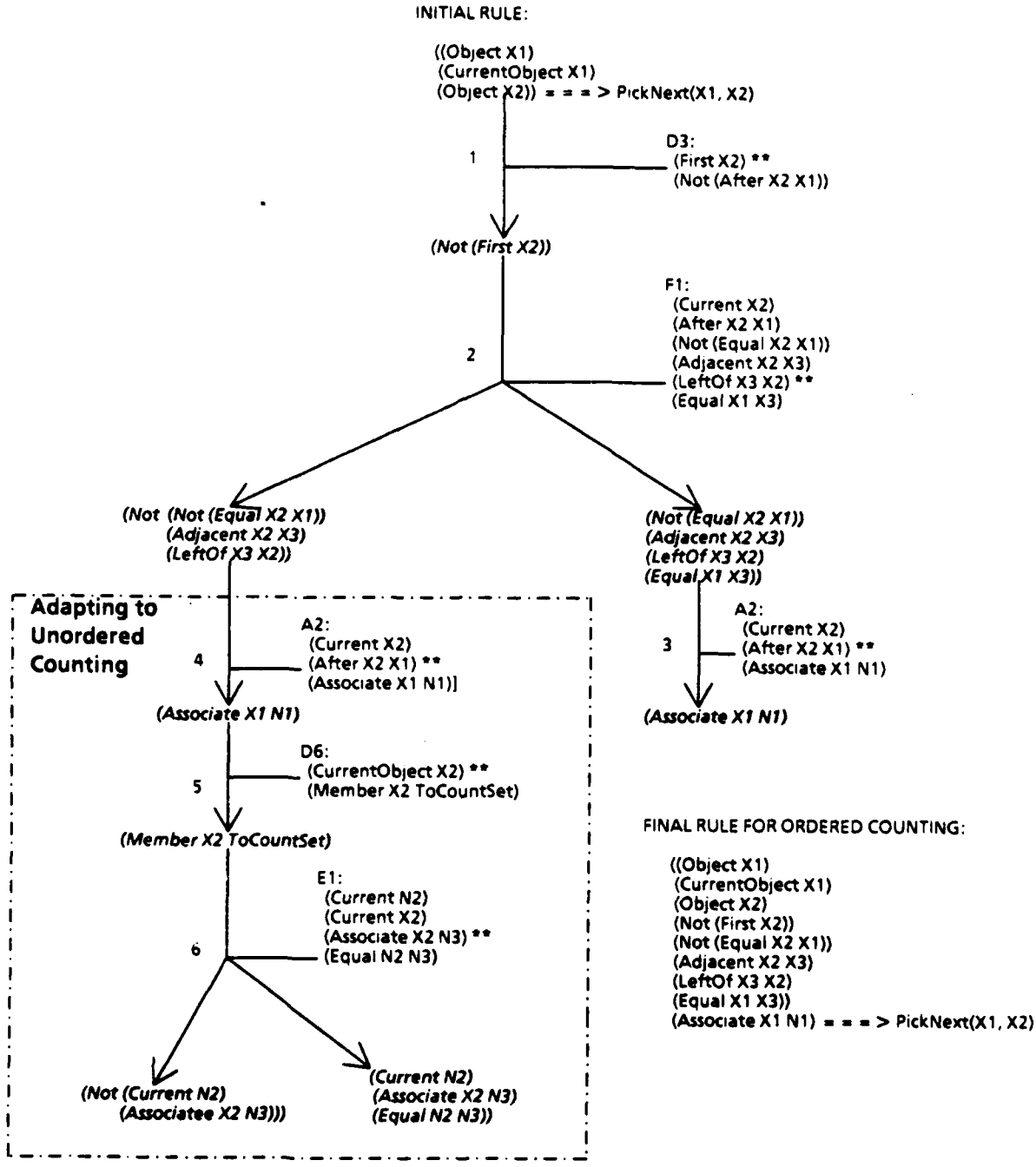


Figure 10: Revisions of rule 2 (see Figure 7) in adaption to unordered counting.

but could conceivably fire in some other, yet-to-be-invented task.

The amount of learning required to adapt from ordered to standard counting is not the same as the amount of learning required to adapt in the other direction. The switch from standard to ordered counting only required two learning steps, one step for each of rule 1 and rule 2, while the switch in the opposite direction requires a total of five learning steps, three for rule 2 (shown in Figure 10) and two for rule 1 (not shown). HS predicts that transfer of training between pairs of tasks is asymmetric.

Transferring from standard to constrained counting

In the task of constrained counting the learner counts an unordered set, but is required to choose objects in such a way that a designated object becomes associated with a designated number. For instance, the learner might be instructed to count in such a way that, say, third object from the left becomes associated with, say, the number five. We present this task to HS by adding the constraints shown in Figure 11. The first constraint represents the general idea that the designated object is associated with the designated number. The two following constraints express the special case of this idea for the initial object and the first number.

F. Designation Constraints

1. Associate the designated object with the designated number.

   (Current X_1)(Designated X_1)(Designated N_1)(After X_1 X_2)(Associate X_2 N_2) ** (Next N_1 N_2)

2. Choose the designated object as the first object only if the designated number is the first number in the number line.

   (Current X)(Designated X)(First X)(Designated N) ** (Origin N)

3. When the designated number is the first number in the number line, and the current object is the first object counted, then it should be the designated object.

   (Current X_1)(First X_1)(Designated X_2)(Designated N_1)(Origin N_1) ** (Equal X_1 X_2)

Figure 11: Constraints that define constrained counting.

As in the previous simulation experiment HS first learned the procedure for standard counting. We then changed the task to constrained counting, and run the system again. Figure 12 shows the effect on the rule for selecting the next object. At the top of the figure is the final rule for unordered counting. As we see the rule violated the first constraint in Figure 11, which leads to the construction of two new rules. In this case, *both* of the new rules are relevant for the task of constrained counting. There is considerable

transfer from one task to the other in this case also, because HS knows, as it were, which rules to revise. As Figure 12 shows, it only requires one learning step to adapt rule 2 to the constrained counting task. It required a total of three learning steps to adapt to constrained counting.

The two demonstrations in this section show that HS can do what Gelman and co-workers have shown that children can do: Adapt a counting procedure to a change in the task demands, rather than having to construct a new procedure from scratch. The pedagogical hope expressed in the Conceptual Understanding Hypothesis is that since children can learn to count with understanding, they might also be able to learn to carry out the symbolic algorithms for arithmetic with understanding. The next question is therefore whether the HS learning mechanism can produce intelligent learning in the domain of symbolic algorithms. This is the topic of the next application.

Correcting errors in a symbolic algorithm

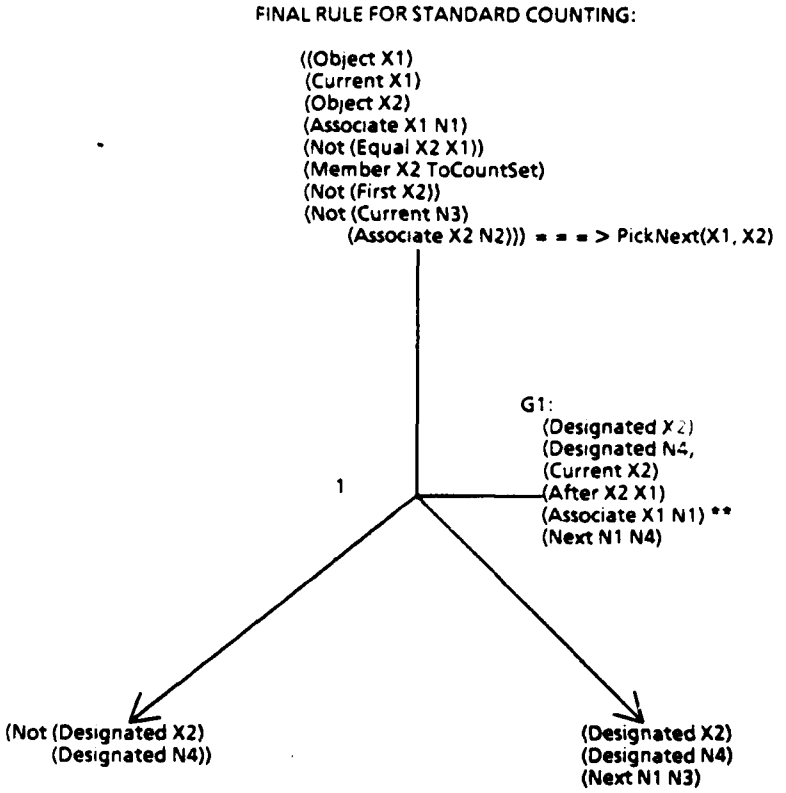
The Conceptual Understanding Hypothesis claims that a learner who constructs a procedure on the basis of principled knowledge is able to spontaneously correct nonsensical errors, without being told what the correct rule is by an outside source, and without having access to a correctly solved example. If the learning of symbolic algorithms such as the subtraction algorithm can proceed in an insightful fashion, the learner should be able to recover from the standard subtraction bugs observed in children's performance. In our third application we show that the HS system can correct errors in a procedure for multi-column subtraction on the basis of knowledge of the principles of subtraction.

In this application HS operates in the standard problem space for subtraction (Ohlsson & Langley, 1985, 1988).[22] A subtraction problem is described in terms of the *values* (v_1, ...) of the digits in the problem, *columns* (d_1, ...), and *rows* (r_1, ...). The columns are numbered from right to left. The initial state contains a description of the spatial layout of the rows and columns, the particular digits of the current problem, a portion of the number line, and the relevant number facts.

There are eleven operators in this problem space: Select a column, move to the next column, decrement a digit, increment a digit, recall the difference between two single digit numbers, recall that the difference between two equal numbers is zero, mark a column as the column to increment, mark a column as the column to decrement, move attention to the left, move attention to the right, and write a digit. The operators for the standard subtraction space is shown in Figure 13 (Part 1 and Part 2). The correct procedure for subtraction with regrouping consists of eleven rules that fire those operations. The state constraints for subtraction that we have developed are inspired by Resnick (1984) and by Resnick and Omanson (1987). We will state each rule and constraint as we analyze each example of how HS learns in this domain. A more detailed description of the subtraction model has been given in a previous report (Ohlsson & Rees, 1987).

Learning experiments in the subtraction domain are not carried out by having HS learn subtraction

[22]We are currently implementing two other representations for subtraction in HS. They will be reported elsewhere.



FINAL RULE FOR CONSTRAINED COUNTING:
(Case 1: Designated Number Occurs Next)

(Object X1)
(Current X1)
(Object X2)
(Associate X1 N1)
(Not (Equal X2 X1))
(Member X2 ToCountSet)
(Not (First X2)
(Not (Current N3)
(Associate X2 N2))
(Not (Designated X2)
(Designated N4)) = = = > PickNext(X1, X2)

FINAL RULE FOR CONSTRAINED COUNTING:
(Case 2: Designated Number Does Not Occur Next)

(Object X1)
(Current X1)
(Object X2)
(Associate X1 N1)
(Not (Equal X2 X1))
(Member X2 ToCountSet)
(Not (First X2))
(Not (Current N3)
(Associate X2 N2))
(Designated X2)
(Designated N4)
(Next N1 N4) = = = > PickNext(X1, X2)

Figure 12: Revisions of rule 2 (see Figure 4) in adaptation to constrained counting.

FirstColumn(C) Takes a column as input and declares that column as the first column.
The addition list is {(**Processing** C)}.
The deletion list is empty.

NextColumn(C_1, C_2)
Takes two columns as inputs, and moves attention from one to the other.
The addition list is {(**Processing** C_2)}.
The deletion list is {(**Processing** C_1)}.

Decrement(R, C_1, C_2, V)
Takes as input the position that is being decremented during a regrouping operation, the position that is being incremented, writes the new value for the decremented digit, and records that the decrement has occured.
The addition list is {(**BorrowedFrom** C_1 **for** C_2)(**CrossedOut** R C_1)(R C_1 **Value** V)}.
The deletion list is {(**BorrowingFrom** C_1 **For** C_2)}.

Increment(R, C, V) Takes as input the position that is being incremented during a regrouping operation, writes the new value for the incremented digit, and records that the increment has occured.
The addition list is {(**Regrouped** C)(**CrossedOut** R C)(R C Value V)}].
The deletion list is {(**Regrouping** C)}.

RecallDiff(V_1, V_2, C)
Takes two numbers and a column as inputs, recalls the difference between the values, and writes the result in the answer-row of the column.
The addition list is {(**AnsRow** C **Value** V_3)}, where $V_3 = V_1 - V_2$.
The deletion list is empty.

Figure 13: Operators for subtraction, Part 1.

from scratch. Instead, we take the correct subtraction procedure and inflict errors of various kinds on it, run HS with the erroneous procedure, and observe whether HS can correct the error or not. We have verified that HS can correct the most frequent errors that have been identified empirically in children's performance. We will illustrate this capability with (a) the SMALLER-FROM-LARGER bug, because it is the most frequent of all bugs, (b) a borrowing bug, because borrowing bugs are the conceptually most difficult bugs, and (c) an ordering bug, because it provides a contrast to the other bugs. More extended examples of learning in the subtraction domain can be found in Ohlsson and Rees (1987).

Recovering from the SMALLER-FROM-LARGER bug

Consider the following faulty rule for subtraction:

> *If c_x is the current column, v_1 is in column c_x, v_1 is in row r_x, v_2 is in column c_x, v_2 is in row r_y, and v_2 is smaller than v_1, then RECALLDIFF(v_1, v_2, c_x).*

The operator RECALLDIFF creates an expression that encodes the retrieved difference, call it v_3, as the

SameDiff(C) Takes a column as input and writes zero in the answer-row for that column.
The addition list is {(**AnswRow C Value 0**)}.
The deletion list is empty.

MarkColumn(C) Takes a column as input, marks it as the column needing to be regrouped.
The addition list is {(**Regrouping C**)}.
The deletion list is empty.

FindColumn(C_1 C_2)
Takes the column to be regrouped and a second column as inputs, and marks the first as the column to be regrouped.
The addition list is {(**BorrowingFrom** C_1 **For** C_2)}.
The deletion list is empty.

ShiftLeft(C_1 C_2 C_3) Takes three columns as inputs, and designates C_1 and C_2 as the columns to be decremented and incremented, respectively.
The addition list is {(**Regrouping** C_1)(**BorrowingFrom** C_3 **For** C_1)}.
The deletion list is {(**Regrouping** C_2)(**BorrowingFrom** C_1 **For** C_2)}.

ShiftRight(C_1 C_2) Takes two columns as inputs, and designates the second one as the one to be incremented.
The addition list {(**Regrouping** C_2)}.
The deletion list is {(**BorrowingFrom** C_1 **For** C_2)}.

WriteValue(C, R, V)
Takes a position, given by a column C and a row R, and a value V as inputs, and writes N in the given position.
The addition list is {(**R C Value V**)}.
The deletion list is empty.

Figure 13: Operators for subtraction, Part 2.

result for column c_x, *i. e.,* O_a *contains the single expression* v_3 *is the result in column* c_x. RECALLDIFF does not delete any expressions, i. e., O_d is empty. This rule ignores the distinction between the minuend and the subtrahend, thus causing the so-called SMALLER/FROM/LARGER bug (Brown & Burton, 1978).

The principle that is violated by the above rule consists of two ideas. First, the purpose of subtraction is to take the subtrahend from the minuend. Second, in the arithmetic of whole numbers, subtraction is undefined when then the minuend is smaller than the subtrahend. The constraint given to HS is:

If row r_{sub} *is the subtrahend row, row* r_{min} *is above* r_{sub}, v_{min} *is in* c_x, v_{min} *is in row* r_{min}, v_{sub} *is in* c_x, v_{sub} *is in row* r_{sub}, v_{min} *is smaller than* v_{sub} *then not(the result in column* c_x *is* v_r*).*

If the minuend in a particular column is smaller than the subtrahend, then there should be no result in that column. It should be noted that the satisfaction pattern is enclosed in a "not" meaning that the constraint is satisfied when the pattern does not match. Also, once the column has been regrouped, the new minuend will not be smaller and this constraint will cease to be relevant.

When applied to the right-most column in, for example, the problem *505 - 19 = ?*, the rule condition R becomes instantiated to

> *c_1 is the current column, 9 is in column c_1, 9 is in row r_{sub}, 5 is in column c_1, 5 is in row r_{min}, and 5 is smaller than 9,*

and the addition list O_a becomes *(4 is the result in column c_1).* The relevance pattern C_r of the constraint becomes instantiated to

> *row r_{sub} is the subtrahend row, row r_{min} is above r_{sub}, 5 is in c_1, 5 is in row r_{min}, 9 is in c_1, 9 is in row r_{sub}, 5 is less than 9,*

and the satisfaction pattern C_s becomes

> *4 is the result in column c_1.*

Since having any result in this column violates the constraint, HS tries to learn from the violation. Obviously, this rule should never fire when the subtrahend is greater then the minuend. To put it another way: If this rule fires when the constraint is relevant, the constraint is guaranteed to be violated. Thus, the rule should fire only when the constraint is not relevant. The learning mechanism does attempt to create two revisions to the rule, but it is successful in only one case.

Revision 1. Ensuring that the constraint is not relevant. First, HS computes (C_r - O_a), using the instantiations of these patterns. However, RECALLDIFF adds only the single expression that matches the satisfaction pattern, so the result is C_r. Next HS removes any parts that are already part of the rule pattern. The result is a single expression which is part of C_r, but not part of either O_a or R: *r_1 is above r_2.* HS replaces the constants r_1 and r_2 with the appropriate variables, and creates a new rule by adding the negation of this expression to the condition of the faulty rule:

> *not* (r_y is above r_x).

This correction cures HS from the SMALLER/FROM/LARGER bug.

Revision 2. Ensuring that the constraint is satisfied. HS computes (C_s - O_a). However the result is empty; RECALLDIFF adds the single expression that matches the satisfaction pattern. The learning mechanism stops at this point and does not attempt to create a second rule.

Recovering from a borrowing bug

The following incorrect subtraction rule finds a column to borrow from when regrouping is needed:

> *If c_x is the column to be regrouped, c_y is a column, v_1 is in column c_y, v_1 is in row r_{min}, row r_{sub} is the subtrahend row, and row r_{min} is above row r_{sub} then FINDCOLUMN(c_y c_x).*

The rule says that *if a particular column needs to be regrouped and there is a second column that*

contains a minuend value, then mark the second column to be borrowed from in order to regroup the first column. FINDCOLUMN adds a single expression representing the fact that c_y is to be borrowed from to regroup c_x. It does not make any deletions. This rule will choose *any* column to borrow from. If, for instance, a particular problem contains three columns, this rule will match three times, once for each column (including the column that is supposed to be regrouped). This rule produces several paths which result in different subtraction bugs. For instance, if the column to the left of the column to be regrouped contains a zero in the minuend, one of the paths will produce the well known BORROW-ACROSS-ZERO bug (Brown & Burton, 1978). This error is produced because this rule attempts to initiate borrowing from all columns. It does not detect the zero and deliberately skip it. Other paths produced by this faulty rule generates other, not necessarily observed, subtraction bugs.

It is possible to apply principled knowledge to this rule to produce a correct rule. The relevant principle states that the column that is borrowed from during regrouping should be just to the left of the column that is being regrouped.[23] This principle is expressed in the following state constraint:

> *If c_x is the column to be regrouped and c_y is the column to borrow from then c_y is to the left of c_x.*

There are two differences that should be noted between this constraint and the previous one. First, the satisfaction pattern is not enclosed inside a "not." Thus, the constraint is satisfied when the satisfaction pattern matches rather than when it does not match. Second, because the rule will fire only when there is a column to regroup and because the operator always adds a column to borrow from, this rule is guaranteed to make the constraint relevant. Thus, the task of learning is to ensure that it will fire only when it will also make the constraint satisfied.

Revision 1. Attempt to ensure that the constraint is not relevant. The difference between the operator's addition and the relevance pattern (C_r - O_a) is: *c_x is the column to be regrouped.* This clause is already part of the rule pattern, however, and adding the negation of it to the rule would produce a new rule that cannot possibly match. Thus, this branch of learning ceases without producing a new rule.

Revision 2. Ensure that the constraint is satisfied. Because the satisfaction pattern and the operator's addition do not overlap, (C_s - O_a) is just C_s. The attempt to compute Revision 1 showed that there is nothing from the relevance pattern to add because (C_r - O_a) is already present in the rule pattern. The revision is to add *c_y is to the left of c_x* to the rule, which produces a correct rule.

---

[23]This particular HS model of subtraction explicitly increments and decrements columns that have zeroes in the top row. In the algorithm taught in schools this process is sometimes abbreviated to *cross out the zero and write a nine, then decrement the next column to the left.*

Recovering from an ordering bug

New rule upon learning, new rules often appear in pairs. One rule of the pair will fire when the particular constraint will not become relevant and the other rule will fire when the constraint will become relevant and satisfied. In the previous two examples, only one of the two revisions succeeded, so only one new rule was created in each learning step. In this final example of error correction in subtraction two new rules are produced.

The relevant rule decides which column to start with in a subtraction problem. It will choose any column, not just the rightmost, i. e., units, column:

If there is no current column and c_x is a column then

FIRSTCOLUMN(c_x)

FIRSTCOLUMN adds *(c_x is the current column)* to working memory and does not delete anything. Like the faulty borrowing rule discussed above, this rule produces branching in the search space. Various odd results are possible along the various branches. For instance, if the rule for choosing the next column to work on correctly chooses the next column to the left, then it might happen that one or more columns to the right are never processed. If the rule for selecting the next column is faulty as well, then columns may be processed in any arbitrary order.

The principle that is violated is that columns should be processed in right to left order. The corresponding state constraint says that if a column is being processed and it is to the left of another column, there should be an answer in that column:

If c_x is the current column and c_x is to the left of c_y then v_r is the result in c_y

This constraint is sufficient to catch both errors in choosing the first column and errors in choosing the next column.

Revision 1. Ensuring that the constraint is not relevant. FIRSTCOLUMN adds the current column so (C_r - O_a) is the second clause in C_r, *c_x is to the left of c_y*. Adding the negation of this expression to the rule produces the obvious requirement that the first column can not be to the left of any other column. This is, of course, the correct rule.

Revision 2. Ensuring that the constraint is satisfied. The satisfaction pattern and the addition do not overlap so (C_s - O_a) is just C_s: *v_r is the result in c_y*. Adding the expression computed for Revision 1 and this expression produces the following rule:

If there is no current column, c_x is a column, c_x is to the left of c_y, and v_r is the result in c_y then FIRSTCOLUMN(c_x)

In the particular representation of subtraction we have chosen for this application, once processing has

started there is always a current column. Thus, it is not possible for there to be no current column and a column with an answer in it at the same time, which means that this rule will never match. Because reasoning about the representation is required to discover that this rule will not match, this conclusion is beyond the power of the learning mechanism, so this rule is added to the rule set. This addition is useless but harmless.

The above examples are simplified in several respects. (a) We usually give HS several deficient initial rules, and we inflect more severe deficiencies on them, so the system starts out with a mixture of different bugs, rather than with a single bug. (b) A severely deficient rule usually violates several constraints, and so has to be revised repeatedly. (c) In order to make the computation of the patterns to be added to faulty rules easier to follow, we have not shown any operators that perform deletions from working memory. (d) For the same reason, we have not shown any constraints that include negated subpatterns. The subtraction model that these examples of error corrections are taken from has been discussed in more detail in Ohlsson & Rees (1987).

Discussion

The behavior of the HS system has several interesting features. First, HS necessarily learns *while* doing. Only by executing its procedure can the system discover that it generates invalid search states. The principled and the procedural knowledge only communicate through the representation for a particular problem situation. Unless the procedure is applied to some problem, there is no way that HS can discover inconsistencies between its procedural and its declarative knowledge. The design of the system is such that HS, like humans, must act in order to learn.

Second, HS is not dependent upon external feedback. It uses its principled knowledge to monitor its own performance, and to discover errors along the path to an answer. It catches itself in mid-air, as it were, learns, and starts over on the current task *before* it reaches an answer. This type of behavior is frequently observed in human learners, but difficult to explain with experience-based learning mechanisms.

Third, HS learns gradually. Rules have to be revised repeatedly. The fact that a rule has been cured from violating one constraint does not guarantee that it will not violate some other constraint. Successive transformations are needed to construct a correct rule even for such a simple task as counting, as the examples above show. Since the learning mechanism works by revising existing rules, the output from one learning step is the input to the next learning step. For HS as for humans, the construction of a new procedure is necessarily a gradual process.

Fourth, the learning mechanism of HS revises a rule by splitting it into two different rules, each version constrained in a different way with respect to the original rule. In most situations only one of those versions is correct from the point of view of the target procedure, and the other other version is a so-called monster rule, i. e., a syntactically correct and executable rule that is not part of the procedure to be learned. In some cases the monster rule can be weeded out on the basis of syntactic criteria, but in many cases it is impossible to decide whether a rule is fruitful or not by inspecting the rule. In those cases both

versions of the rule are executed in future trials, and *HS gets rid of the monster rule by further learning.* The monster rules are executed and constrained repeatedly, until they are so constrained that they cannot match any search state. They are then harmless and have, functionally speaking, been deleted.[24] If we think of HS' learning as a search through the procedure space, we can describe this phenomenon by saying that HS does not have a criterion for when it has reached the goal state, i. e., a correct procedure. Therefore, it has to continue searching in order to verify that there are no further improvements to make.

The fact that HS weeds out monster rules by further learning constitutes a prediction that human learners will continue to make mistakes even after they have acquired the correct rules for a procedure. The reason is that they have not yet learned to ignore the alternative, incorrect rules that were constructed in the same learning step as the correct rule. Further practice is necessary in order to get rid of those rules. Hence, HS predicts that practice will be beneficial for some period of time over and above what is needed in order to reach correct performance. This point illustrates well the complex interactions between knowledge and practice. It also illustrates the necessity of implementing and running information processing models. The result that further practice is necessary even after the correct rule has been constructed is a rather complicated, and unanticipated, prediction from our theory that we almost certainly would not have discovered without computer implementation of the theory.

Fifth, HS can transfer a procedure from one task to another. The flexibility of HS' procedure for counting *does not reside in the final procedure that HS learns.* The set of final production rules learned by HS is, taken by itself, as brittle a procedure as any other. It is only when those rules are executed *in the context of the state constraints* that flexibility is achieved. The flexibility of HS does not reside in the *type* of procedure it learns, or in the problem solving method embodied in that procedure, but in the fact that the procedure is executed within a cognitive context that includes principled knowledge of the task environment.

Sixth, HS finds it easier to transfer between tasks in one direction than in the other. For instance, the learning process that transforms a procedure for unordered counting into a procedure for ordered counting is not the same as the process that transforms a procedure for ordered counting into a procedure for unordered counting. Depending upon which constraints are violated, the number of learning steps involved in adapting from one task to another may be different from the number of learning steps required to adapt in the opposite direction. This constitutes a prediction that transfer of training between pairs of tasks is asymmetric.

Seventh, learning in HS consists of a transition from a knowledge-based to a procedure-based performance. In the initial phase of learning, the system makes much use of its principled knowledge, because the grossly incomplete procedure makes errors at every step. As the procedure is gradually

[24]We could model actual deletion of such rules by assigning weights to rules, and postulating (a) that the weight decays over time unless the rule is fired, and (b) that rules with a weight below some threshold value is purged from the system. We have not implemented such a mechanism in the current version of the HS model.

completed, fewer and fewer of the steps are incorrect, and the state constraints kick into action less and less. At the end of learning, the state constraints have dropped out of sight completely, because the production rules now generate only correct solutions. If we assume that the state constraints have levels of activation and that the activation level is a function of how often the constraint is violated, then HS models the transition from mindful action, in which all steps are thought about in relation to the system's principled knowledge, to routine action, in which an already mastered procedure is simply run off, as it were, without much thought. The principled knowledge of HS only plays a role in its performance when something goes wrong, i. e., some inconsistency between the current state of the world and its knowledge is detected. In short, HS only thinks, as it were, about the current problem when it is forced to do so by some difficulty.

Eight, adaptation to a new task involves revision of those rules that are not appropriate for the new task. Rules that are inappropriate will be revised, because they will violate some constraint for the new task. Rules that are appropriate for the new task will not be revised, since they do not cause any constraint violations. Hence, by construction, HS knows which parts of a procedure to retain and which to revise when faced with a change in the task demands. Like humans, HS can build on what it has previously learned when learning a new procedure.

Relations to Previous Research

The purpose of this section is to outline the major conceptual differences between the HS model and other computational models of the acquisition of arithmetic procedures. To the best of our knowledge, there are only two previous analyses of the problem of deriving arithmetic procedures from principled knowledge, both of which make use of so-called planning nets, but neither of which resulted in an implemented simulation model (VanLehn & Brown, 1980; Greeno, Riley, & Gelman, 1984; Smith, Greeno, & Vitolo, in press). We also know of two efforts to simulate human procedure acquisition in arithmetic which employ experience-based, rather than knowledge-based, learning methods (VanLehn, 1983a, 1983b, 1985a, 1985b, 1986; Neches, 1981, 1982, 1987; Neches & Hayes, 1978).

Planning net analyses of arithmetic procedures

VanLehn and Brown (1980) have pointed out that a program for a procedure does not reveal the purpose of that procedure. Programs and flow diagrams specify the steps of a procedure and the conditions under which those steps are to be carried out, but they do not describe the reason why a particular step is included in the procedure, or why it is executed under those conditions. For instance, the procedure for carrying in multi-column addition can be described as follows: *when the sum of column n is larger than nine, then detach the units part of the sum, record that part as the result for column n, and add the remaining part to the column to the left.* But this description does not reveal that the purpose of the carrying operation is to make sure that each exponent of ten is represented by a single-digit coefficient in the answer. VanLehn and Brown (1980) introduce the term "telelogical semantics" to refer to a description of the purpose of the steps in a procedure.

Drawing upon A. I. analyses of planning, VanLehn and Brown (1980) proposed a methodology for generating a procedure from a goal in such a way that the trace of the generation constitutes a teleological semantics for the procedure. Their methodology assumes that planning begins with a goal, a set of operations, a set of planning heuristics, and a characterization of a problem situation. Planning begins by posing the goal, and proceeds by expanding it, i. e., replacing it with a structure consisting of subgoals and/or executable operations. Each step in the process is guided by a planning heuristic. The process continues until all goals have been expanded into executable operations, and the execution of the procedure does not contradict any features of the problem situation. The trace of a planning process consists of a graph in which the nodes are (partial) procedures, i. e. procedures that contain yet-to-be-expanded subgoals. The links between the nodes are labelled with the planning heuristic that led from one procedure to the next. VanLehn and Brown (1980) call such a trace a *planning net.*

VanLehn and Brown (1980) invented planning nets in order to compare procedures with respect to closeness or similarity. They found that program-level representations of procedures do not yield reasonable similarity metrics: A minor conceptual change in a procedure can give raise to huge differences in the program for that procedure. They propose a closeness metric based on the planning net representation that does reproduce intuitive judgments about similarity between procedures. They use the metric to discuss the pedagogical merit of concrete models for arithmetic such as Dienes blocks,

and to design a sequence of concrete models for instruction in subtraction (VanLehn & Brown, 1980, pp. 132-136). The planning mechanism is not implemented as a computer program. They do not claim that the process of deriving a planning net for an arithmetic procedure correspond to the mental process of someone who is trying to learn that procedure.

The idea of deriving a procedure by successively expanding goals into operations within the constraints imposed by a particular problem situation was taken up by Greeno and co-workers in their theory of counting competence (Greeno, Riley, and Gelman, 1984; Riley & Greeno, 1980; Smith & Greeno, 1983; Smith, Greeno, & Vitolo, in press). The basic assumption of their theory is that knowledge of principles is encoded in *action schemata*. A schema is an action described at a high level of abstraction. The description includes both inputs (prerequisites), success criteria (postrequisites), outputs (consequences and effects), and conditions that have to remain true during the execution of the action (corequisites). For instance, the following schema describes the action of picking up an object:

PICK-UP(*a*)
Prerequisities: *movable*(*a*);
empty(*Hand*).
Consequences: *in*(*a*, *Hand*).

The PICK-UP schema says that the prerequisites for picking up an object *a* are that *a* is movable and that one's hand is empty. The consequence of picking up an object is that the object is in the hand (Greeno, Riley, & Gelman, 1984, p. 105). The PICK-UP schema is an example of a schema that can be executed without expansion into other schemata. Executable schemata correspond to what is called operators in most computational models of problem solving.

Knowledge of the counting principles is encoded in a total of twelve different action schemata, most of them considerably more complicated than the PICK-UP schema. A central schema is the description of the action of mapping a set onto a subset of another set:

MATCH(*X*, *Y*)

Prerequisities: *empty*(*A*);
empty(*B*).
Corequisities: *subset*(*A*, *X*), where *A* = {*x*: *tagged*(*x*)};
subset(*B*, *Y*), where *B* = {*y*: *used*(*y*)};[25]
equal(*A*, *B*).
Postrequisities: For all *x*, *member*(*x*, *X*) --> *member*(*x*, *A*).
Consequence: *equal*(*X*, *B*).

The MATCH schema says that in order to match a set X to a set Y, we must first have an empty subset of each set. We then act on those subsets (in some manner that is not specified in the schema itself) until the subset A of X becomes equal to X itself. We cause A to grow, as it were, until it includes all of X. While doing this, we make sure (in some yet-to-be-specified way) that it always remains the case that

[25]The two properties *tagged* and *used* serve bookkeeping purposes in the Greeno et. al (1984) analysis of counting.

the subset B of Y has the same number of members as A, i. e, we cause B to grow at the same rate as A. The result of acting in this way is that when A includes all of X, X is guaranteed to have the same number of elements as B. Since B is a subset of Y, X has thereby been mapped onto Y. The MATCH schema is part of the encoding of the one-to-one mapping principle (Greeno, Riley, & Gelman, 1984, p. 113). It is an example of a non-executable schema; it cannot be executed as it stands, but has to be expanded into executable schemata.

The computational mechanism postulated in the action schema theory is a planning mechanism that bears a family resemblance to the type of mechanism sketched by VanLehn and Brown (1980). It takes as inputs the goal of deciding the cardinality of a set, the collection of twelve action schemata, and a setting. The setting describes the problem and the physical situation in which the problem is to be solved. The planning mechanism consists of two components. The first component is a mechanism for *backward chaining* that matches the goal against the consequences of the action schemata.[26] Schemata that can satisfy that goal are posed as potential actions in the plan. The prerequisites of those schemata are then posed as subgoals. This process continues until all goals are satisfied either by the setting or by the consequences of executable schemata that are included in the plan. The second component of the planning mechanism is a *theorem prover* that decides whether a particular pre-, co-, or postrequisite is satisfied in a particular setting by trying to prove that requisite as a theorem.

The trace of the planning mechanism is a graph that Greeno et. al (1984) call a *planning net*, with reference to the work by VanLehn and Brown (1980). However, there is little formal resemblance between the two types of graphs. The planning nets of VanLehn and Brown (1980) have partial procedures as nodes. Links are labelled with planning heuristics. The label H on the link from node N to node M means that applying planning heuristic H to procedure N yields procedure M (see Figure 18.2, VanLehn and Brown, 1980, p. 115). In contrast, the planning nets in Greeno et. al (1984) have action schemata, tests, and goals as nodes, and the links are labelled as pre-, post, or co-requisities. The meaning of, say, the prerequiste link R from, say, action schema node A to, say, goal node G is that obtainment of goal G satisfies prerequisite R for action A (see Figure 4, Greeno et. al, 1984, p. 119). The two types of graphs, although formally different, share the purpose of explaining a procedure by relating steps to goals.

The main phenomenon investigated by Greeo et. al (1984) is the flexibility[27] of childrens' counting performances, in particular, the fact that children can adapt their counting procedures to a variety of settings. Flexibility of performance is explained in the action schema theory by the fact the planning mechanism can derive different procedures for different settings from one and the same set of action

---

[26]Greeno, Riley, & Gelman (1984, p. 116-117) incorrectly describe their mechanism as a form of means-ends analysis. However, means-ends analysis consists of computing a difference between a goal and a situation, and retrieving an operator that can reduce that difference from a difference-operator table (Ernst & Newell, 1969). The mechanism described in Greeno et. al (1984) does not compute differences, and does not make use of a difference-operator table.

[27]Greeno et. al (1984, p. 122) make a distinction between *flexibility* and *robustness*. This distinction is not necessary for the discussion here, so we use the term "flexibility" to cover both concepts.

schemata. The planning mechanism does not appear to have any resources for making use of the procedure for one setting in the derivation of a procedure for another setting; each procedure is derived *de novo*.

Since both the state constraint theory proposed here and the action schema theory by Greeno and co-workers address the same psychological phenomenon, it ought to be possible to make a detailed comparison between them with respect ability to account for data, clarity, simplicity, generality, etc. However, such a comparison is complicated by the fact that the action schema theory is not proposed as a process theory, but as a competence theory. Greeno et. al explicitly reject any claims about the psychological reality of the planning mechanism that they describe:

> We note that we do not necessarily identify the *process of derivation* of planning nets as a plausible psychological hypothesis. As with other hypotheses about competence, we restrict our claim to psychological reality to the *content* of the knowledge that is attributed to individuals and to the *structures* that are implied by that knowledge. In our analysis, the relation between competence and performance structures has the form of derivations in which the performance structures are consequences of competence structures, derived by a planning system. However, we have not tried to determine the form of the dependence between competence and performance structures in human cognition.
>
> (Greeno, Riley, & Gelman, 1984, p. 104)

> We consider the content of the competence in our analysis a plausible set of hypotheses about children's tacit knowledge, but the way in which the three components of competence are used in deriving planning nets should be interpreted as a formal relation, not necessarily corresponding to cognitive mechanisms.
>
> (Greeno, Riley, & Gelman, 1984, p. 138)

In short, the action schema theory spells out the rational connections between the counting principles, encoded as action schemata, and the procedures that generate counting behavior, but the planning process that generates those connections does not (necessarily) correspond to any mental process.

If the computational machinery of the action schema theory is not to be interpreted as a psychological hypothesis, what are the empirical claims of the theory? In what respects can the theory be compared to a process model such as the HS system? In the two excerpts quoted above Greeno et. al claim that children know the content of the action schemata. But the action schemata are supposed to encode the counting principles, so this claim appears, at first glance, as a mere restatement of the conclusion by Gelman and Gallistel (1978) that children know those principles.

However, inspection of the action schemata does not support the idea that they are nothing but an encoding of the counting principles. For instance, the MATCH schema (see above) can be paraphrased as saying that *if an initially empty subset A of a set X is changed so as to include more and more of X, and if an initially empty subset B of an other set Y is changed so as to always have the same size as A, then, when A has become identical to A, B will have the same size as X.* This is a rather complicated

set-theoretic theorem that cannot reasonably be said to be included among the counting principles. The claim that children have the knowledge encoded in the action schemata is therefore a claim that they know the counting principles *plus the other principles embedded in those schemata*. But the authors do not specify those other principles.

Identification of which principles are encoded in the action schemata is further complicated by the fact that principles are spread out among the schemata, and that children are not hypothesized to either know or not know the principles:

> We did not formulate a schema for understanding of order, another schema for one-to-one correspondance, and so on. Instead, it seemed more reasonable to hypothesize schemata that represent different aspects of the various principles, and often include aspects of different principles. If our analysis is accepted, then competence for each of the principles is distributed among several schemata, rather than being located in any single structure. This emphasizes that a child should not be considered as either having or not having competence regarding any of the principles ... .
>
> (Greeno, Riley, & Gelman, 1984, p. 137)

Even if we had a list of the principles encoded in the action schemata, the evaluation of the claim that children know those principles would still be problematic. The action schemata are hypothesized to be known *implicitly* (Greeno, Riley, & Gelman, 1984, pp. 106 and 137). Hence, the claim cannot be tested by interviewing children directly about the content of the schemata. Knowledge of the schemata must be inferred from observations of performance. But we do not know what to look for in children's performance, since the action schema theory does not claim psychological reality for its process mechanisms.

However, the action schema theory can be interpreted as making a different kind of empirical claim, although it is not stated explicitly by Greeno et. al (1984). The authors draw an analogy between their work and the chomskyan methodology for competence theories in the study of syntax. Strict interpretation of this analogy implies that we can assign a psychological interpretation to the *set of all counting procedures* that can be generated from the action schemata with the described planning mechanism. The theory can be interpreted as claiming that the action schemata and the planning mechanism generate all counting procedures that competent number users would judge as correct,[28] a claim that is, in principle, empirically testable, and which can be used to compare the action schema theory to other theories. For example, it would be interesting to compare the set of counting procedures that can be generated by action schema theory with the set of counting procedures that can be learned by the HS system. Greeno et. al (1984, pp. 137-138) mention the possibility of deriving such a prediction from their theory, but they do not develop it, with the motivation that there is no characterization of the set of all possible procedures, analogous to the characterization of the set of all possible strings of symbols in a language.

[28] An alternative interpretation is that they generate all counting procedures that humans can learn.

A major difficulty in the evaluation of the implicit claim that the set of procedures that can be generated from action schema theory coincides with the set of correct counting procedures is that the planning mechanism postulated in that theory is not fully specified.[29] The backward chaining mechanism is given an informal specification that appears precise enough to support implementation (Greeno et. al, 1984, p. 116-117). However, it is radically incomplete: Greeno et. al does not deal with the issue of how to order sibling subgoals, one of the central problems for planning mechanisms. Ordering subgoals is crucial for the derivation of even the simplest procedure. The authors themselves express doubts as to the sufficiency of the computational mechanism they describe (see Greeno et. al, 1984, p. 116, footnote 7, and p. 122). Furthermore, the theorem prover that decides whether requisites are satisfied is not described, even in outline. It is supposed to have access to inference rules and general propositions. An example of a general proposition is that *objects in a straight line can be ordered, starting at one end and proceeding to the other* (Greeno et. al, 1984, p. 118). The relation between general propositions and action schemata is not clarified. Without a fully specified computational mechanism, the set of procedures that can be derived from the action schemata is not well defined.

In summary, the planning net analyses of arithmetic procedures by VanLehn and Brown (1980) and by Greeno et. al (1984) are based on the notion of constructing a procedure by successively expanding a goal into a plan for how to achieve that goal. However, the analysis by VanLehn and Brown (1980) is not intended as a psychological theory, but is aimed at the definition of a similarity metric for procedures. The action schemata theory of Greeno et. al (1984) is a competence theory, and the empirical claims of the theory are unclear. Neither analysis has been embodied in an implemented system that can generate runnable procedures.

Simulation models of empirical learning in arithmetic

Neches (1981, 1982, 1987) has described the *Heuristic Procedure Modification* system (HPM), a production system architecture of learning based on the idea that significant improvements in a procedure can be computed by noticing patterns in the internal trace of the procedure, patterns that indicate some labour-saving transformation of the procedure is possible. The HPM system is based on a typology of strategy transformations that eliminates redundancies, produces shortcuts, replaces one method with a computationally more efficient method, etc. (Neches & Hayes, 1978). For instance, if a procedure uses a partial result at two different points in a computation, that procedure can be improved by storing that result when it is first computed, and retrieving it, rather than recomputing it, when it is needed the second time. In order to support the detection of the triggering patterns for these strategy transformations, the HPM architecture stores a very detailed trace of the execution of a procedure. For every expression that is written into working memory, information is stored about the production rule that was responsible for the

[29]The implementation status of the action schema theory of counting competence is somewhat complicated. The first published account of the theory (Greeno, Riley, & Gelman, 1984) mentions an implemented performance model for counting, described in detail in Riley and Greeno (1980). This performance model, called "SC" for "Standard Counting", consists of 54 ACTP production rules that count correctly in four different settings. However, the computational mechanism that is supposed to derive those rules from action schemata was not, according to the authors, implemented (footnote 8, Greeno, Riley, & Gelman, 1984, p. 116). More recent publications (Smith, Greeno, & Vitolo, in press; Smith & Greeno, 1983) mention an implementation of the planning mechanism in the PRISM production system language. However, no technical details are given in these publications.

creation of that expression, the conditions that led to the firing of that rule, the goal that was active when the rule was fired, etc. Each strategy transformation mechanism inspects this trace for the occurrence of the type of redundancy that it is designed to deal with, and transforms the procedure accordingly.

The major phenomenon explained by the HPM system is the discovery of the so-called MIN-strategy for simple addition. There is evidence that children who are taught to solve simple addition problems by combining the sets corresponding to the two addends and then counting the combined set quickly realize that they can proceed more efficiently by initializing their counting with the larger addend, and then counting only the elements in the smaller set (Groen & Resnick, 1977). The HPM application to this phenomenon shows how the relevant strategy transformation can be achieved through the elimination of redundancy (Resnick & Neches, 1984). For instance, HPM notices that in counting the combined set, the number corresponding to the larger addend is generated *en route* to the answer. Since that number can always be retrieved from the problem statement, it is redundant to re-compute it. Hence, the counting can begin with the larger addend. The HPM system explains procedure acquisition through the application of content-independent mechanisms to a trace of a procedure. It does not explain the role and function of general knowledge in procedure acquisition.

VanLehn (1983a, 1985a, 1985b) has described Sierra, a procedure induction system that can generate a subtraction procedure from a set of solved examples. The main phenomenon explained by Sierra is the multitude of bugs in children's performance on multi-column subtraction problems. The Sierra system outputs a set of procedures in response to a sequence of solved examples. One explanatory principle of VanLehn's theory is that procedure induction is an intrinsically hard problem; indeed, some induction problems are known to be unsolvable. As a result, a procedure induced from solved examples can be expected to be incomplete. Incomplete procedures may lead to *impasses*, situations in which the procedure either cannot determine the next step, or finds that the preconditions for the next step are not satisfied. A second explanatory principle of VanLehn's theory is that the learner will deal with impasses by making local changes to his/her procedure (Brown & VanLehn, 1980, 1982). He has identified a small set of general transformations, called *repairs*, that a learner can apply to a procedure in order to break out of an impasse. For instance, an impasse can be repaired by skipping the step that cannot be carried out, or by replacing it by another step. If the repairs are applied to the procedures generated by Sierra the result is a set of buggy algorithms, i. e., algorithms that solve subtraction problems, but solve them incorrectly. The Sierra model plus the theory of repairs explain a significant proportion of the subtraction bugs that have been observed in the performance of school children. Neither the procedure induction mechanism nor the repair mechanism make use of arithmetic principles.

The theory of Kurt VanLehn and co-workers is the dual of the theory produced here. They assume that school children do not, in fact, consult principled knowledge of arithmetic in the construction of arithmetic procedures, but learn them by rote. The goal of their theory is to provide a computational model of rote learning, and thereby explain the actual behavior of school children. Mathematics educators, on the other hand, assume that school children could, in principle, learn arithmetic procedures

in a meaningful way. The goal of the state constraint theory is to provide a computational model of meaningful learning, and thereby illustrate the desired behavior of school children. Obviously, these two research efforts, although based on opposite hypotheses about learning, are complementary rather than contradictory.

Discussion

The computer simulation technique is applied to educationally relevant task domains with increasing frequency (Ohlsson, 1988a). However, in spite of this fact, and in spite of the large amount of research devoted to the psychology and pedagogy of elementary arithmetic, only three computational models of the acquisition of arithmetic procedures have been proposed prior to the work reported here. The two process models--the strategy transformation model by Robert Neches and the procedure induction/repair model by Kurt VanLehn--both use experience-based learning techniques, and hence do not address the question of the role and function of principled knowledge in procedural learning. The action schema theory of counting competence does address the problem of principled knowledge, but it has not been embodied in a runnable system that can generate behavioral predictions.

The learning of arithmetic procedures is a complex process that is unlikely to have a simple explanation. Each of the theories reviewed address a different aspect of arithmetic learning. A complete model of arithmetic learning would presumably be able to plan, to detect and correct mistakes, to detect and eliminate redundancies, to induce procedures from examples, as well as to repair a procedure in order to break out of an impasse. The action schema theory, state constraint theory, strategy transformation theory, and the procedure induction theory, and the repair theory are complementary research efforts.

Other research efforts have addressed the issue of the role and function of principled knowledge in procedure acquisition. Anderson (1982, 1983a, 1983b, 1986) have proposed the mechanism of proceduralization, in which a declarative principle, e. g., a geometric theorem, is stepwise contextualized and converted into procedural form. Ohlsson (1987b) has proposed a related model that specifies the conditions under which it is meaningful to apply proceduralization. Both of these theories assume that declarative principles occur as data-elements in working memory. The psychological interpretation of this is that principles are known explicitly rather than implicitly. This assumption is plausible with respect to domains like high school geometry and Lisp programming, but not with respect tc the domain of counting. Hagert (1986) has proposed a methodology for deriving procedures from abstract specifications which bears a family resemblance to the planning net analyses, but which uses the methodology of logic programming. Procedures are derived from abstract specifications through a deductive argument. Principles of the domain appear as premises in the derivation. The notion of deriving a principle from an abstract specification has also been investigated in software engineering (see, e. g., Balzer, 1985). Finally, Artificial Intelligence research has invented the technique of explanation-based learning, in which principled knowledge is used to construct an explanation why an example is an instance of a particular concept. By collapsing the explanation into a single rule, a general recognition rule for that concept is created without any need to consult further examples (DeJong & Mooney, 1986; Mitchell, Keller, & Kedar-

Cabelli, 1986). These learning techniques have not been applied to arithmetic.

General Discussion

The first subsection below summarizes the argument we have been making. In the following subsection we state the strengths of the state constraint theory and of the HS model. Finally, we go on to describe the major weaknesses of our theory, and the problems they pose for future work.

Summary

The research problem addressed in this report is the problem of the function of conceptual understanding in performance and learning, with special emphasis on arithmetic learning. Mathematics educators have proposed the Conceptual Understanding Hypothesis, which claims that if children knew the concepts and principles of arithmetic, acquisition of computational algorithms would proceed smoothly. If children understood what they are doing, this hypothesis claims, they could discover procedures on their own, learned procedures would be flexible, nonsensical errors would be corrected spontaneously, and learned procedures would easily combine to form higher-order procedures. The major example of knowledge-based procedure aquisition in arithmetic is the domain of counting.[30] Empirical studies have shown that children know the principles of this domain, that they can construct correct and general procedures for counting without formal instruction, and that the learned procedures are flexible. The pedagogical hope expressed in Conceptual Understanding Hypothesis is that if we teach children the conceptual basis of the arithmetic procedures, then the acquisition of those procedures will proceed in the same insightful fashion.

Evaluation of the Conceptual Understanding Hypothesis requires explicit hypotheses about (a) what is meant by understanding, the content of understanding, and how that content is represented in human memory, and about (b) the computational mechanisms by which understanding influences performance and procedure acquisition. The theory proposed in this report is based on the idea that understanding enables the learner to notice and correct his/her own mistakes. According to this theory understanding consists of principles that constrain the possible problem states. The principles can guide performance, because the system tries to avoid solution paths that violate them. Furthermore, the principles can guide procedure acquisition, because the particular *way* in which a procedure violates a principle contains information about how that procedure should be revised.

We implemented the theory in a production system architecture called HS. The structure of HS corresponds closely to the structure of heuristic search. Production rules correspond to search heuristics, and working memory correspond to the current search state. HS takes one step through the problem space during each cycle of operation. The major innovation of the model is the augmentation of these mechanisms with the state constraint representation of principled knowledge. We represent principles as ordered pairs of patterns, where the *relevance* pattern circumscribes the set of situations in which a principle is relevant, and the *satisfaction* pattern is a criterion which a situation has to satisfy in order to be consistent with the principle. The two patterns are matched against search states with the same pattern

[30] As we noted on page 5, this conclusion is not shared by all researchers in the field.

matcher that matches the conditions of production rules against working memory. The state constraints influence performance in that the number of constraint violations serves as a cost variable in the evaluation function for search states. The state constraints influence learning in that the HS learning algorithm reacts to a constraint violation by replacing the faulty rule with two other rules, constrained so as to avoid producing similar constraint violations in future applications.

We reported three applications of the HS system. The first two applications reproduce the major phenomena with respect to children's counting: Children can construct a correct and general counting procedure without formal instruction in counting, and they can adapt the procedure to changes in the task. The third application investigated the behavior of HS in the domain of multi-column subtraction. If HS is given a subtraction procedure that suffers from one or more subtraction bugs, it can correct those bugs without external feedback, given a state constraint representation of the concepts and principles of subtraction. These three applications constitutes a substantiation of the Conceptual Understanding Hypothesis: a learning system that can acquire a counting procedure in an insightful way has been demonstrated to be capable of learning in the domain of multi-column subtraction as well.

The HS learning algorithm is a rational learning technique, because it derives a procedure from knowledge rather than from experience. Rational learning processes have not been widely studied in cognitive psychology, and there are few theoretical efforts to clarify them. The analyses most relevant to our work are the planning net analyses by VanLehn and Brown (1980) and by Greeno, Riley, and Gelman (1984). However, neither of these analyses attempted to provide a process model of procedure acquisition, and neither resulted in an implemented system. There have been other attempts to formulate learning mechanisms that make use of a declarative representation of domain knowlege, but they have not been applied to arithmetic. The process models of procedure acquisition in arithmetic that have been proposed are models of experience-based, rather than knowledge-based, learning. The HS system goes beyond previous theoretical efforts in that it presents an implemented process model of knowledge-based procedure acquisition in arithmetic.

Strengths of the state constraint theory

The state constraint theory provides interesting and novel answers to several difficult questions with respect to the relation between understanding and performance. It also generates qualitative predictions which are, in principle, empirically testable. Finally, the state constraint theory fares well on other evaluation criteria such as generality and parsimony.

Interpretation of meaningful learning

What is the difference between solving a problem correctly but blindly, and solving that same problem correctly and with understanding? According to the state constraint theory, there is no difference *in the procedure being executed* in the two situations. A procedure is just a set of dispositions to act in certain ways under certain circumstances; it cannot be either blind or intelligent, only more or less efficient. The theory says that understanding is present when the procedure is executed *in the context of the learner's world knowledge.* Thoughtful execution consists of matching the outcomes of the procedural steps

against the concepts and principles of the relevant domain. Thoughtless execution, on the other hand, consists of doing the steps without reflection on their outcomes. Hence, the exhortation "think about what you are doing!" is slightly off-target; according to the state constraint theory, the better advice is "think about the *results* of what you are doing!".

What is the nature of knowledge? Discussions about this question usually assume that principles are either descriptive (e. g., "All swans are white") or predictive (e. g., "The sun will rise tomorrow"). The state constraint theory claims that neither of these two models of principled knowledge is essential for procedure acquisition. Instead, principled knowledge consists of *constraints* on the possible states of affairs (e. g., "You cannot withdraw more money than you have in your account bank"). Conservation laws in physics, e. g., the principle that energy cannot be destroyed or created, are examples of constraints, as are arithmetic principles, e. g., the laws of commutativity and associativity.

What function does knowledge have in performance? What good does it do to think about the results of what you are doing, and how are constraint principles helpful? For every procedure there will exist situations in which that procedure is applicable, but in which it will not produce desirable results. Intelligent behavior therefore depends on the ability to imagine the outcomes of actions, and to weed out the mistaken actions before they are carried out. The function of principled knowledge is to enable a person to catch and correct the mistakes that his/her procedure--any procedure--will unavoidably make when confronted with unfamiliar situations.

This interpretation of the function of knowledge solves two technical problems that other accounts of the function of knowledge have been unable to deal with. The first problem concerns the effect of adding more knowledge to the system. Humans perform and learn better and faster the more they know. But all computational mechanisms for using knowledge proposed to date suffer from combinatorial explosions: The more knowledge the mechanism is provided with, the slower it will work and the less likely it is to behave intelligently. For instance, the more action schemata the planning mechanism of Greeno, Riley, and Gelman (1984) is supplied with, the harder the planning problem, because the more alternatives have to be considered at each point in the planning process. In general, mechanisms that combine knowledge units into larger structures cannot explain why people function better, the larger their knowledge base. However, according to our theory, state constraints are not combined with each other. Each state constraint is matched against the current search state independently of the other constraints. Hence, there is no combinatorial explosion as the number of knowledge items grows.[31]

The second problem that state constraint theory deals successfully with concerns the effect of partial knowledge. Human beings operate very well with partial knowledge; in fact, they hardly ever operate in any other way. But most computational mechanisms for using principled knowledge cannot function if their knowledge base is incomplete. This is a serious problem with, for instance, explanation-based

[31]The amount of computation required to match constraints against states grows with the number of knowledge items, but the growth need not be exponential, or even linear (Forgy, 1982).

learning (DeJong & Mooney, 1986; Mitchell, Keller, & Kedar-Cabelli, 1986). In general, techniques that build larger knowledge structures out of smaller units--an explanation, a plan, a proof, etc.--cannot proceed if one of the units is missing. But the state constraints in our theory are not combined into higher-order structures. If one of the constraints is missing, the system becomes less constrained, and it will therefore have to search more. But the power of the other constraints to guide performance and learning is not affected. We have verified that HS can use a partial set of state constraints to guide its performance on subtraction problems.

What is the nature of the change that occurs during meaningful procedure acquisition? The state constraint theory claims that the essence of learning with understanding is that structure is transferred from declarative to procedural knowledge.[32] When a person first confronts an unfamiliar problem situation he/she needs to think hard about it, because almost every action generates constraint violations. As the procedure is gradually corrected, the state constraints need to kick into action less and less often; execution of the procedure can be removed from reflection and becomes more mechanical. Finally, when the procedure is correct, there is no need to consult the state constraints in order to execute it. Hence, the acquisition of a procedure is a process of moving from acting under the influence of knowledge to "just doing it", as common sense would have it.

In summary, the state constraint theory locates understanding in the cognitive context within which a procedure is executed, it assumes that knowledge constrains the possible states of affairs, and it claims that the function of understanding is to enable the learner to catch and correct his/her mistakes. The theory explains why the cognitive machinery does not suffer from combinatorial explosion as the number of knowledge items grows, but on the contrary becomes more efficient. It also explains why humans can operate well with partial knowledge. These two phenomena pose major difficulties for other computational models of understanding. Finally, the theory explains the passage from reflection to action during meaningful learning, because it claims that the learner only consults his/her knowledge, as it were, when something goes wrong.

Qualitative predictions about behavioral phenomena

The state constraint theory makes four qualitative predictions about human behavior. First, the theory predicts that additional learning is required after the correct rules for a particular task have been discovered. The reason is that the learning mechanism creates rules in pairs, each member of the pair constraining the parent rule in a different way. When the correct rule is created, another, probably incorrect, companion rule is therefore created also. At the time of creation, it is impossible to know which of the two rules is the correct rule. HS can only identify the correct rule by evoking the rules and observe their effects. HS gets rid of the incorrect rule by constraining it until it is over-constrained, and cannot fire. Hence, additional learning trials are necessary after the correct rule has been created in order to get rid of the superfluous companion rule. Those learning trials will generate errors. Hence, the theory predicts that

[32] In computer science terms, structure is transferred from the *test* (the knowledge) to the *generator* (the procedure).

errors will *necessarily* occur after the correct rule has been discovered.

The second prediction of the HS system concerns the interaction between knowledge and performance. Since state constraints guide performance by assigning a cost to a search state that violates a principle, it is possible for HS to produce incorrect solutions in the presence of a complete set of constraints. It turns out that incorrect solution paths in the subtraction domain are shorter than correct paths. Hence, if the cost of a constraint violation is less than the cost of taking an extra step, HS prefers the shorter path, even though it violates one or more constraints. We have verified that if HS is given an incomplete subtraction procedure but a complete set of principles, it produces incorrect answers on some subtraction problems for some settings of the cost parameters.

The third prediction derived from the state constraint theory concerns the level of difficulty of learning a particular procedure. The theory predicts that a procedure will be easy to learn to the extent that each step in the procedure has results that can be judged for correctness on the basis of the principles of the domain. Counting is easy to learn according to this theory, because every step in counting either follows or violates the one-one mapping principle. Mistakes are therefore immediately detectable by someone who knows the one-one mapping principle. A procedure is hard to learn to the extent that it contains a large number of steps that are not on the correct solution path, but which nevertheless are consistent with all the principles of that domain that the learner knows. In short, state constraint theory makes the counterintuitive prediction that the *larger* the number of constraints that have to be satisfied by a particular procedure, the easier that procedure is to acquire.

The fourth prediction that we discovered in the simulation runs is that the amount of cognitive effort required to switch from task *A* to task *B* is not the same as the cognitive effort required to switch from task *B* to task *A*. If HS learns to count objects in arbitrary order, it can learn to take a pre-defined order into account in a single learning step. However, if it initially learns to count objects in a particular order, learning to count objects when that order is not present requires several learning steps. This amounts to a prediction that transfer between tasks will be greater in one direction than in the other. Such asymmetry in transfer between related tasks is intuitively plausible.

Other evaluation criteria

The state constraint theory is *well integrated into current cognitive theory*. The theory is an extension of the major hypothesis about problem solving to emerge in the past decades, namely that problem solving consists of heuristic search, carried out by a production system architecture. The HS model is build out of off-the-shelf computational mechanisms that have already been proven fruitful in explaining a wide range of cognitive phenomena. Although the simulation runs analyzed in this report are from the domain of arithmetic, the state constraint theory is nevertheless a *general* theory. The computational mechanisms of heuristic search and of production system architectures are formulated in domain-independent terms. They are not limited to arithmetic but can, in principle, be applied to any task domain. The mechanism of matching state constraints against search states and counting the number of constraint violations is a general mechanism, not limited to arithmetic. The mechanism for revising rules

in response to constraint violations is also formulated in domain-independent terms. The state constraint theory postulates a *simple* computational mechanism. The constraints are compared to search states with the same pattern matcher that compares production rules to search states. Hence, no new major computational mechanisms had to be invented in order to augment the standard theory of problem solving with the state constraint representation.

Weaknesses and future directions

The state constraint theory errs by being incomplete. There are several aspects of procedure acquisition that it does not deal with, among them the role of experience, procedural errors, remote errors, undetected errors, and the hierarchical organization of cognitive skills.

The state constraint theory as embodied in the HS simulation model does not explain the *function of experience* in the learning of procedures. While the experience-based learning models for arithmetic proposed by Neches (1981, 1982, 1987) and by VanLehn (1983a, 1983b, 1985a, 1985b, 1986) contain no mechanisms by which principled knowledge can influence procedure acquisition, the HS model errs in the opposite direction. It contains no mechanism by which procedures can be created by storing and generalizing steps that experience has shown give the right results. HS *only* learns by deriving procedures from its knowledge. But human beings obviously learn both by applying their knowledge and by generalizing from experiences. The state constraint theory is therefore radically incomplete. It does not describe how experience-based learning happens, nor how empirical and rational learning mechanisms collaborate in the creation of procedures.

The lack of experience-based learning mechanisms prevents HS from handling purely *procedural errors*, i. e., errors that cannot be described as violations of the principles of the relevant domain. Such errors will occcur under two circumstances. First, in the case of incomplete principled knowledge, there might be errors that can, in principle, be described as principle violations, but which the system cannot, in fact, so describe, because it does not know the relevant principle. Second, in some domains there might be steps which are not on the correct solution path, but which are not incorrect in the sense of violating any domain principle. For instance, in mathematical proof tasks there are a large number of proof paths which are valid, but which do not lead to the target theorem. The learning mechanism that we have implemented for the HS model cannot correct such errors.

The state constraint theory is also unable to deal with *remote errors*. The assumption that all errors violate principles of the domain implies a simple solution to the assignment of blame problem. If all errors violate constraints, then it is always the last rule to fire before an error is detected that needs to be revised. If, however, there are errors that do not violate constraints, then those errors will not be detected at the time they are made. But they could cause constraint violations several steps later. In that case the faulty rule fired several steps before the step in which the error was detected, and identifying the rule that is responsible for the error is a difficult problem.

The HS model is an idealization in the sense that it does not suffer from *undetected errors*; it is guaranteed to discover every violation of its constraints. Clearly, people often fail to detect the errors they

make. This phenomenon can be modeled in HS by assigning a probability to the pattern matching process that compares search states with constraints. The system would then make errors that it could, in principle, detect, but which would, in fact, go undetected on some proportion of the trials in which they occur. There are two reasons why we have not implemented such a mechanism in the current version of HS. The first reason is that the structure of the HS architecture implies that if the detection of constraint violations is probabilistic, so is the matching of production rules: both processes are carried out by the same pattern matching mechanism. Production systems with probabilistic rule matching have not been explored, and nothing is known about how to program them.[33] Hence, such a step is major theoretical move which is not immediately related to our main objective of understanding the role and function of principled knowledge in procedure acquisition. The second reason is that little is gained by introducing quantitative parameters without independent empirical grounding at this stage in the development of the model.

The state constraint theory is also incomplete in that it does not deal with the *hierarchical organization* of procedural skills. The HS learning algorithm does not create hierarchically organized procedures. As a consequence the state constraint theory cannot explain why understanding facilitates the combination of already learned procedures into higher-order procedures, which is one of the effects hypothesized by adherents of the Conceptual Understanding Hypothesis. In contrast,both the planning mechanism proposed by Greeno and co-workers (Greeno, Riley, and Gelman, 1984; Smith, Greeno, & Vitolo, in press) deals readily with the hierarchical organization of cognitive skills, as does the model of procedure induction proposed by VanLehn (1983a).

The weaknesses of the state constraint theory stem from its exclusive focus on errors that violate principles of the domain. Future work will extend the knowledge-based learning mechanism described in this report with one or more experience-based learning mechanisms. There are many ways to combine experience-based and knowledge-based learning. For example, one possibility is to combine a planning mechanism like the one proposed by Greeno et. al (1984) with the state constraint mechanism. Such a system would learn by constructing an initial procedure through planning, and then revise it in the course of execution if it turns out to violate principles of the domain. Many other hypotheses are possible. We do not yet have any conclusions as to which type of combination of experience-based and learning-based learning mechanisms is most likely to predict the details of human behavior.

Future work will move from a concern with explaining qualitative features of human behavior, such as the ability to adapt a procedure to changes in the relevant task, to a concern with quantitative predictions. We can, in principle, derive quantitative predictions from the current version of the HS model. For instance, by running HS repeatedly on the task of learning to count, we can generate predictions about

[33]Probabilistic matching is related to, but not identical with, *partial* matching (Langley, 1983a, p. 291). In partial matching only a part of a rule pattern has to match in order for a rule to fire. In probabilistic matching a rule will only fire on some proportion of the cycles in which its rule pattern did match completely.

the frequency distribution of error types[34] at different levels of learning. Deriving such predictions would be premature at the present stage of development of the model.

One might object to the work reported here that the most radical weakness in the state constraint theory is that it does not explain where principled knowledge comes from in the first place. However, this objection represents a misunderstanding of the problem we set out to solve. We have tried to formulate a theory of how knowledge of principles, once acquired, can be used in the learning of procedures; we have not tried to explain the acquisition of principles. This way of proceeding seemingly presupposes that the principles of a particular task domain can be known before one knows how to act in that domain, an intuitively implausible idea.[35] However, the state constraint theory does not require that *all* principles are known before procedural learning starts. This is an ideal case only. In a real learning situation we would expect the learning of principles and the learning of procedures to be interleaved.

The idealization that all principles are acquired before procedural learning starts is appropriate for the work reported here, because our goal was to clarify the *nature* of the link between understanding and procedure acquisition hypothesized in the Conceptual Understanding Hypothesis. The pedagogical hope expressed in that hypothesis is precisely that conceptual understanding can be the *basis* for procedure acquisition. The state constraint theory is one explanation of how access to conceptual understanding can enable a learner to discover arithmetic procedures, adapt procedures to changes in the task environment, and self-correct nonsensical errors. Future work will address the question of how principles are acquired.

What are the instructional implications of the state constraint theory? Suppose, for the sake of the discussion, that we decide to adopt the theory in its current form, without augmentation with additional learning mechanisms. The theory then implies that a procedure cannot be taught by describing the steps in the procedure to the learner. There are no mechanisms in HS that can make use of an instruction like "first you do X, then you Y". In particular, the state constraint theory implies that it is not useful to tell a learner who just committed a mistake what the correct action would have been. The theory implies that instruction should focus on the state of the problem, not on the learner's actions. In correcting an error the instructor should help the learner to focus on the problem, and to see what is wrong with its current state, reminding him/her of the principles of the domain, if necessary. The instructor should not tell the learner what he/she should have done to avoid the error, but describe which state the problem ought to be in, and leave it to the learner to figure out what action or actions would achieve that state. We are not proposing that mathematics teachers revise their instruction in according with these implications. We are not ready to derive specific recommendations for teaching from our theory until the theory has been subject to stringent empirical tests. These admittedly speculative comments are ment to illustrate that

---

[34]There are four types of errors in counting: skipping an object, counting an object repeatedly, skipping a number, using a number repeatedly, answering without having counted all objects, and continuing to generate numbers after all objects have been counted, thus answering with too high a number.

[35]Notice that Gelman and Meck (1983) have argued on the basis of extensive empirical studies that this is, in fact, the case in the domain of counting.

idealized computational theories of the function of conceptual understanding in the learning of arithmetic procedures can generate rather specific implications for instruction.

References

Adelson, B., Burstein, M., Gentner, D., Hammond, K., Holyoak, K., Thagard, P. (1988). The role of analogy in a theory of problem-solving. *The Tenth Annual Conference of the Cognitive Science Society* (pp. 298-304). Montreal, Quebec, Canada: Erlbaum.

Anderson, J. R. (1976). *Language, memory, and thought.* Hillsdale, NJ: Erlbaum.

Anderson, J. R. (1981). *Cognitive skills and their acquisition.* Hillsdale, NJ: Erlbaum.

Anderson, J. R. (1982). Acquisition of cognitive skill. *Psychological Review, 89*, 369-406.

Anderson, J. R. (1983a). Acquisition of proof skills in geometry. In R. S. Michalski, J. G. Carbonell, & T. M. Mitchell (Eds.), *Machine learning: An artificial intelligence approach.* Palo Alto, CA: Tioga Publishing Co.

Anderson, J. R. (1983b). *The architecture of cognition.* Cambridge, MA: Harvard University Press.

Anderson, J. R. (1986). Knowledge compilation: The general learning mechanism. In R. S. Michalski, J. G. Carbonell, & T. M. Mitchell (Eds.), *Machine learning: An artificial intelligence approach* (Vol. II, pp. 289-310). Los Altos, CA: Morgan Kaufmann Pub., Inc.

Anderson, J. R., Greeno, J. G., Kline, P. J., & Neves, D. M. (1981). Acquisition of problem-solving skill. In J. R. Anderson (Ed.), *Cognitive skills and their acquisition.* (pp. 191-230). Hillsdale, NJ: Erlbaum.

Anzai, Y., & Simon, H. A. (1979). The theory of learning by doing. *Psychological Review, 86*, 124-140.

Ashcraft, M. H. (1983). *Simulating network retrieval of arithmetic facts* (Tech. Report No. 1983/10). Pittsburgh, PA: University of Pittsburgh, Learning Rese ırch and Development Center.

Balzer, R. (1985). A 15 year perspective on automaticv programming. *IEEE Transactions on software engineering, SE-11*(11), 1257-1268.

Baroody, A. J., & Ginsburg, H. P. (1986). The relationship between initial meaning and mechanical knowledge of arithmetic. In J. Hiebert (Ed.), *Conceptual and procedural knowledge: The case of mathematics.* (pp. 75-112). Hillsdale, NJ: Erlbaum.

Bell, A. W., Costello, J., & Kuchemann, D. E. (1983). *A review of research in mathematical education. Part A. Research on learning and teaching.* Berks, U.K.: NFER-Nelson.

Bolc, L. (Ed.). (1987). *Computational models of learning.* Berlin, Federal Republic of Germany: Springer-Verlag.

Brainerd, C. J. (1979). *The origins of the number concept.* New York, NY: Praeger.

Briars, D., & Siegler, R. (1984). A featural analysis of preschoolers' counting knowledge. *Developmental Psychology, 20*, 607-618.

Brooks, L. W., & Dansereau, D. F. (1987). Transfer of information: An instructional perspective. In S. M. Cormier, & J. D. Hagman (Eds.), *Transfer of learning: Contemporary research and applications.* New York: Academic Press.

Brown, J. S., & Burton, R. R. (1978). Diagnostic models for procedural bugs in basic mathematical skills. *Cognitive Science, 2*, 155-192.

Brown, J. S., & VanLehn, K. (1980). Repair theory: A generative theory of bugs in procedural skills.

Cognitive Science, 4, 379-426.

Brown, J. S., & VanLehn, K. (1982). Towards a generative theory of "bugs". In T. P. Carpenter, J.M. Moser, & T. A. Romberg (Eds.), *Addition and subtraction: A cognitive perspective* (pp. 117-135). Hillsdale, NJ: Erlbaum.

Burton, R. (1982). Diagnosing bugs in a simple procedural skill. In D. Sleeman & J. S. Brown (Eds.), *Intelligent tutoring systems* (pp. 157-183). London: Academic Press.

Carbonell, J. G. (1982). Learning by analogy: Formulating and generalizing plans from past experience. In R. S. Michalski, J. G. Carbonell & T. M. Mitchell (Eds.), *Machine learning: An artificial intelligence approach* (pp. 85-111). Palo Alto, CA: Tioga Press.

Carbonell, J. G. (1983). Derivational analogy and its role in problem solving. *Proceedings of the National Conference on Artificial Intelligence* (64-69). Washington, D. C.

Clark, K., & Taernlund, S. A. (Eds.). (1982). *Logic programming.* London: Academic Press.

Davis, R. B. (1984). *Learning mathematics. The cognitive science approach to mathematics education.* Norwood, NJ: Ablex.

Davis, R. B., & King, J. (1976). An overview of production systems. In E. W. Elcock & D. Michie (Eds.), *Machine intelligence* (Vol. 8). New York: John Wiley.

DeJong, G., & Mooney, R. (1986). Explanation-based learning: An alternative view. *Machine Learning, 1*, 145-176.

Egan, D. E., & Greeno, J. G. (1973). Acquiring cognitive structure by discovery and rule learning. *Journal of Educational Psychology, 64*, 85-97.

Ernst, G. W., & Newell, A. (1969). *GPS: A case study in generality and problem solving.* New York: Academic Press.

Feynman, R. (1965). *The character of physical law.* Cambridge, MA: The M.I.T. Press.

Forgy, C. L. (1982). Rete: A fast algorithm for the many pattern/many object pattern match problem. *Artificial Intelligence, 19*, 17-37.

Fuson, K. C., & Hall, J. W. (1983). The acquisition of early number word meanings: A conceptual analysis and review. In H. P. Ginsburg (Ed.), *The development of mathematical thinking* (pp. 49-107). London, UK: Academic Press.

Galambos, J. A., Abelson, R. P., & Black, J. B. (Eds.). (1986). *Knowledge structures.* Hillsdale, NJ: Erlbaum.

Gelman, R., & Gallistel, C. R. (1978). *The child's understanding of number.* Cambridge, MA: Harvard University Press.

Gelman, R., & Meck, E. (1983). Preschoolers' counting: Principle before skill. *Cognition, 13*, 343-359.

Gelman, R., & Meck, E. (1986). The notion of principle: The case of counting. In J. H. Hiebert (Ed.), *Conceptual and procedural knowledge: The case of mathematics* (pp. 29-57). Hillsdale, NJ: Erlbaum.

Gelman, R., Meck, E., & Merkin, s. (1986). Young children's numerical competence. *Cognitive Development, 1*, 1-29.

Gentner, D. (1987). *Analogical inference and analogical access* (Report No. UIUCDCS-R-87-1365). Urbana: University of Illinois at Urbana-Champaign, Dept. of Computer Science.

Greeno, J. G. (1978). Understanding and procedural knowledge in mathematics instruction. *Educational Psychologist, 12*, 262-283.

Greeno, J. G. (1983). Forms of understanding in mathematical problem solving. In S. G. Paris, G. M. Olson & H. W. Stevenson (Eds.), *Learning and motivation in the classroom* (pp. 83-111). Hillsdale, NJ: Erlbaum

Greeno, J. G., Riley, M. S., & Gelman, R. (1984). Conceptual competence and children's counting. *Cognitive Psychology, 16*, 94-143.

Groen, G., & Resnick, L. (1977). Can preschool children invent addition algorithms? *Journal of Educational Psychology, 69*(6), 645-652.

Groner, R., Groner, M., & Bischof, W. F. (Eds.). (1983). *Methods of heuristics.* Hillsdale, NJ: Erlbaum.

Hagert, G. (1986). Logic modeling of conceptual structures: Steps towards a computational theory of reasoning (Doctoral dissertation, Uppsala University, 1986). *Uppsala Thesis in Computing Science, 3/86.*

Hayes-Roth, F., Klahr, P., & Mostow, D. J. (1981). Advice taking and knowledge refinement: An iterative view of skill acquisition. In J. R. Anderson (Ed.), *Cognitive skills and their acquisition* (pp. 231-253). Hillsdale, NJ: Erlbaum.

Hiebert, J. (Ed.). (1986). *Conceptual and procedural knowledge: The case of mathematics.* Hillsdale, NJ: Erlbaum.

Hiebert, J., & Wearne, D. (1986). Procedures over concepts: The acquisition of decimal number knowledge. In J. Hiebert (Ed.), *Conceptual and procedural knowledge: The case of mathematics* (pp. 199-223). Hillsdale, NJ: Erlbaum.

Hinton, G. E. (1987). *Connectionist learning procedures* (Tech. Report No. CMU-CS-87-115). Pittsburgh, PA: Carnegie-Mellon University, Dept. of Computer Science.

Holland, J. H., Holyoak, K. J., Nisbett, R. E., & Thagard, P. R. (1986). *Induction: Processes of inference, learning, and discovery.* Cambridge, MA: The MIT Press.

Holyoak, K. J. (1984). Analogical thinking and human intelligence. In R. J. Sternberg (Ed.), *Advances in the psychology of human intelligence* (Vol. 2). Hillsdale, NJ: Erlbaum.

Jones, R., & Langley, P. (1988). A theory of scientific problem solving. *The Tenth Annual Conference of the Cognitive Science Society* (pp. 244-250). Montreal, Quebec, Canada: Erlbaum.

Katona, G. (1967). *Organizing and memorizing: Studies in the psychology of learning and teaching.* New York: Hafner Pub. Co.

Klahr, D., Langley, P., & Neches, R. (Eds.). (1987). *Production system models of learning and development.* Cambridge, MA: The MIT Press.

Kotovsky, K., Hayes, J. R., & Simon, H. A. (1985). Why are some problems hard? Evidence from tower of Hanoi. *Cognitive Science, 17*, 248-294.

Laird, J. E., Rosenbloom, P. S., & Newell, A. (1986). *Universal subgoaling and chunking: The automatic generation and learning of goal hierarchies.* Boston, MA: Kluwer.

Langley, P. (1983a). Exploring the space of cognitive architectures. *Behavior Research Methods & Instrumentation, 15*(2), 289-299.

Langley, P. (1983b). Learning search strategies through discrimination. *International Journal of Man-Machine Studies, 18*, 513-541.

Langley, P. (1985). Learning to search: From weak methods to domain-specific heuristics. *Cognitive Science, 9*, 217-260.

Langley, P. (1987). A general theory of discrimination learning. In D. Klahr, P. Langley, & R. Neches (Eds.), *Production system models of learning and development* (pp. 99-161). Cambridge, MA: The MIT Press.

Langley, P., Wogulis, J., & Ohlsson, S. (in press). Rules and principles in cognitive diagnosis. In N. Frederiksen (Ed.), *Diagnostic monitoring of skill and knowledge acquisition*. Hillsdale, NJ: Erlbaum.

Mayer, R. E., Stiehl, C. C., & Greeno, J. G. (1975). Acquisition of understanding and skill in relation to subjects' preparation and meaningfulness of instruction. *Journal of Educational Psychology, 67*, 331-350.

Michener, E. R. (1978). Understanding understanding mathematics. *Cognitive Science, 2*, 361-383.

Mitchell, T. M., Keller, R. M., & Kedar-Cabelli, S. T. (1986). Explanation-based generalization: A unifying view. *Machine Learning, 1*, 47-80.

Moore, J. & Newell, A. (1974). How can Merlin understand? In L. W. Gregg (Ed.), *Knowledge and cognition*. Potomac, MD: Earlbaum.

Neches, R. T. (1981). A computational formalism for heuristic procedure modification. *Proceedings of the Seventh International Joint Conference on Artificial Intelligence* (pp. 283-288). Vancouver, British Columbia, Canada.

Neches, R. (1982). Simulation systems for cognitive psychology. *Behavior Research Methods & Instrumentation, 14*(2), 77-91.

Neches, R. (1987). Learning through incremental refinement of procedures. In D. Klahr, P. Langley, & R. Neches (Eds.), *Production system models of learning and development*. Cambridge, MA: The MIT Press.

Neches, R., & Hayes, J. R. (1978). Progress towards a taxonomy of strategy transformations. In A. M. Lesgold, J. W. Pellegrino, S. Fokkema & R. Glaser (Eds.), *Cognitive psychology and instruction*. New York: Plenum Books.

Neches, R., Langley, P., & Klahr, D. (1987). Learning, development, and production systems. In D. Klahr, P. Langley, & R. Neches (Eds.), *Production system models of learning and development* (pp. 1-53). Cambridge, MA: The MIT Press.

Neimark, E. R., & Estes, W. K. (Eds.). (1967). *Stimulus sampling theory*. San Francisco, CA: Holden-Dag.

Neves, D. M., & Anderson, J. R. (1981). Knowledge compilation: Mechanisms for the automatization of cognitive skills. In J. R. Anderson (Ed.), *Cognitive skills and their acquisition* (pp. 57-84). Hillsdale, NJ: Erlbaum.

Newell, A. (1966). *On the analysis of human problem solving protocols* (Research Grant No. MH-07722-22). Pittsburgh, PA: Carnegie-Mellon University, Dept. of Computer Science.

Newell, A. (1972). A theoretical exploration of mechanisms for coding the stimulus. In A. W. Melton & E. Martin (Eds.), *Coding processes in human memory.* Washington, D. C.: Winston.

Newell, A. (1973). Production systems: Models of control structures. In W. G. Chase (Ed.), *Visual information processing* (pp.463-526). New York: Academic Press.

Newell, A. (1980). Reasoning, problem solving and decision processes: The problem space as a fundamental category. In R. Nickerson (Ed.), *Attention and performance VIII.* Hillsdale, NJ: Erlbaum.

Newell, A., & Simon, H. A. (1959). *The simulation of human thought* (Report No. P-1734). Santa Monica, CA: The Rand Corporation, Mathematics Division.

Newell, A., & Simon, H. A. (1972). *Human problem solving.* Englewood Cliffs, NJ: Prentice-Hall, Inc.

Nilsson, N. J. (1971). *Problem-solving methods in artificial intelligence.* New York: McGraw-Hill.

Ohlsson, S. (1983). A constrained mechanism for procedural learning. *Proceedings of the Eighth International Joint Conference on Artificial Intelligence* (pp. 426-428). Karlsruhe, Federal Republic of Germany.

Ohlsson, S. (1986). Rational vs. empirical learning. In H. J. Kugler (Ed.), *Information Processing* (p. 841). North, Holland: Elsevier Science Publishers.

Ohlsson, S. (1987a). Transfer of training in procedural learning: A matter of conjectures and refutations? In L. Bolc (Ed.), *Computational models of learning* (pp. 55-88). Berlin, Federal Republic of Germany: Springer-Verlag.

Ohlsson, S. (1987b). Truth versus appropriateness: Relating declarative to procedural knowledge. In D. Klahr, P. Langley, & R. Neches (Eds.), *Production system models of learning and development* (pp. 287-327). Cambridge, MA: The MIT Press.

Ohlsson, S. (1988). Computer simulation and its impact on educational research and practice. *International Journal of Educational Research, 12,* 5-12. Oxford: Pergamon Press.

Ohlsson, S., & Langley, P. (1985). *Identifying solution paths in cognitive diagnosis* (Tech. Report CMU-RI-TR-85-2). Pittsburgh, PA: Carnegie-Mellon University, The Robotics Institute.

Ohlsson, S., & Langley, P. (1988). Psychological evaluation of path hypotheses in cognitive diagnosis. In H. Mandl, & A. Lesgold (Eds.), *Learning issues for intelligent tutoring systems.* New York: Springer-Verlag.

Ohlsson, S., & Rees, E. (1987). *Rational learning: Deriving arithmetic procedures from state constraints* (Tech. Report No. KUL-87-04). Pittsburgh, PA: University of Pittsburgh, Learning Research and Development Center.

Patel, V. L., & Groen, G. J. (1986). Knowledge based solution strategies in medical reasoning. *Cognitive Science, 10*(1) 91-116

Pearl, J. (1984). *Heuristics: Intelligent search strategies for computer problem solving.* Reading, MA: Addison-Wesley Pub. Co.

Piaget, J. (1959). *The child's conception of number.* London, UK: Routledge & Kegan Paul.

Resnick, L. B. (1983). A developmental theory of number understanding. In H.P. Ginsburg (Ed.), *The development of mathematical thinking* (pp.109-151). New York: Academic Press.

Resnick, L. B. (1984). Beyond error analysis: The role of understanding in elementary school arithmetic. In H. N. Cheek (Ed.), *Diagnostic and prescriptive mathematics: Issues, ideas, and insights* (pp. 2-14). Kent, OH: Research Council for Diagnostic and Prescriptive Mathematics.

Resnick, L. B., & Ford, W. W. (1981). *The psychology of mathematics for instruction*, Hillsdale, NJ: Erlbaum.

Resnick, L. B., & Neches, R. (1981). Factors affecting individual differences in learning ability. In R. J. Sternberg (Ed.), *Advances in the psychology of human intelligence* (Vol. 2, pp. 275-323). Hillsdale, NJ: Erlbaum.

Resnick, L. B., & Omanson, S. F. (1987). Learning to understand arithmetic. In R. Glaser (Ed.), *Advances in instructional psychology* (Vol. 3, pp. 41-95). Hillsdale, NJ: Erlbaum.

Riley, M. S., & Greeno, J. G. (1980). *Details of programming a model of children's counting in ACTP* (Tech. Report No. 6). Pittsburgh: University of Pittsburgh, Learning Research and Development Center.

Romberg, T. A. & Carpenter, T. P. (1986). Research on teaching and learning mathematics: Two disciplines of scientific inquiry. In M. C. Wittrock (Ed.), *Handbook of research on teaching* (3rd ed., pp. 850-873). New York: MacMillan.

Rumelhart, D. E., & Norman, D. A. (1981). Analogical processes in learning. In J. R. Anderson (Ed.), *Cognitive skills and their acquisition* (pp. 335-359). Hillsdale, NJ: Erlbaum.

Schank, R. C. (1986). *Explanation patterns. Understanding mechanically and creatively.* Hillsdale, NJ: Erlbaum.

Schoenfeld, A. H. (1985). *Mathematical problem solving.* Orlando, FL: Academic Press.

Silver, E. A. (Ed.). (1985). *Teaching and learning mathematical problem solving: Multiple research perspectives.* Hillsdale, NJ: Erlbaum.

Silver, E. (1986). Using conceptual and procedural knowledge: A focus on relationships. In J. Hiebert (Ed.), *Conceptual and procedural knowledge: The case of mathematics* (pp. 181-198). Hillsdale, NJ: Erlbaum.

Smith, D. A., & Greeno, J. G. (1983). *Implicit understanding and competence: A theoretical analysis of procedural and utilizational competence.* Paper presented at the meeting of the Cognitive Science Society, Rochester, NY.

Smith, D. A., Greeno, J. G., & Vitolo, T. M., (in press). A model of competence for counting. *Cognitive Science.*

VanLehn, K. (1983a). *Felicity conditions for human skill acquisition: Validating an AI-based theory* (Tech. Report No. CIS-21). Palo Alto, CA: Xerox PARC.

VanLehn, K. (1983b). On the representation of procedures in repair theory. In H. P. Ginsburg (Ed.), *The development of mathematical thinking* (pp. 201-252). New York: Academic Press.

VanLehn, K. (1985a). *Acquiring procedural skills from lesson sequences* (Tech. Report No. ISL-9). Palo Alto, CA: Xerox PARC.

VanLehn, K. (1985b). *Learning one subprocedure per lesson* (Tech. Report No. ISL-10). Palo Alto, CA: Xerox PARC.

VanLehn, K. (1985c). *Theory reform caused by an argumentation tool* (Tech. Report No. ISL-11). Palo

Alto, CA: Xerox PARC.

VanLehn, K. (1986). Arithmetic procedures are induced from examples. In J. H. Hiebert (Ed.), *Conceptual and procedural knowledge: The case of mathematics* (pp. 133-179). Hillsdale, NJ: Erlbaum.

VanLehn, K., & Brown, J. S. (1980). Planning nets: A representation for formalizing analogies and semantic models of procedural skills. In R. E. Snow, P. A. Federico, & W. E. Montague (Eds.), *Aptitude, learning, and instruction: Vol. 2. Cognitive process analyses and problem solving* (pp. 95-137). Hillsdale, NJ: Erlbaum.

VanLehn, K., Brown, J. S., & Greeno, J. (1982). *Competitive argumentation in computational theories of cognition* (Report No. CIS-14). Palo Alto, CA: Xerox PARC.

Winograd, T. (1975). Frame representations and the declarative/procedural controversy. In D. G. Bobrow & A. Collins (Eds.), *Representation and understanding. Studies in cognitive science* (pp. 185-210). New York: Academic Press.

Young, R. M., & O'Shea, T. (1981). Errors in children's subtraction. *Cognitive Science, 5*, 153-177.

KUL Reports

1985

Ohlsson, S., & Langley, P. (April, 1985). Psychological evaluation of path hypotheses in cognitive diagnosis (Technical Report No. 1985/2). Pittsburgh: Learning Research and Development Center, University of Pittsburgh.

1986

Ohlsson, S. (January, 1986). Some principles of intelligent tutoring (Technical Report No. 1986/2). Pittsburgh: Learning Research and Development Center, University of Pittsburgh.

Ohlsson, S. (June, 1986). Computer simulation and its impact on educational research and practice (Technical Report No. 1986/14). Pittsburgh: Learning Research and Development Center, University of Pittsburgh.

Ohlsson, S. (October, 1986). Sense and reference in the design of interactive illustrations for rational numbers (Technical Report No. 1986/18). Pittsburgh: Learning Research and Development Center, University of Pittsburgh.

1987

Ohlsson, S. (April, 1987). A semantics for fraction concepts (Technical Report No. KUL-87-01). Pittsburgh: Learning Research and Development Center, University of Pittsburgh.

Ohlsson, S. (September, 1987). Trace analysis and spatial reasoning: An example of intensive cognitive diagnosis and its implications for testing (Technical Report No. KUL-87-02). Pittsburgh: Learning Research and Development Center, University of Pittsburgh.

Ohlsson, S., Nicholas, S., & Bee, N. (December, 1987). Interactive illustrations for fractions: A progress report (Technical Report No. KUL-87-03). Pittsburgh: Learning Research and Development Center, University of Pittsburgh.

Ohlsson, S., & Rees, E. (December, 1987). Rational learning: Deriving arithmetic procedures from state constraints (Technical Report No. KUL-87-04). Pittsburgh: Learning Research and Development Center, University of Pittsburgh.

1988

Ohlsson, S. (February, 1988). Mathematical meaning and applicational meaning in the semantics for fractions and related concepts (Technical Report No. KUL-88-01). Pittsburgh: Learning Research and Development Center, University of Pittsburgh.

Ohlsson, S. (March, 1988). Principled understanding of addition and subtraction (Technical Report No. KUL-88-02). Pittsburgh: Learning Research and Development Center, University of Pittsburgh.

Ohlsson, S., & Rees, E. (August, 1988). An information processing analysis of the function of conceptual understanding in the learning of arithmetic procedures (Technical Report No. KUL-88-03). Pittsburgh: Learning Research and Development Center, University of Pittsburgh.

ONR Distribution List

ACKERMAN PHILLIP L
Dr. Phillip L. Ackerman
University of Minnesota
Department of Psychology
75 East River Road
N218 Elliott Hall
Minneapolis, MN 55455

AFHRL/MPD
Air Force Human
Resources Lab
AFHRL/MPD
Brooks, AFB, TX 78235

AFOSR LIFE SCIENCES
AFOSR,
Life Sciences Directorate
Bolling Air Force Base
Washington, DC 20332

AHLERS ROBERT
Dr. Robert Ahlers
Code N711
Human Factors Laboratory
Naval Training Systems Center
Orlando, FL 32813

ANDERSON JOHN R
Dr. John R. Anderson
Department of Psychology
Carnegie-Mellon University
Schenley Park
Pittsburgh, PA 15213

ARI TECHNICAL DIRECTOR
Technical Director, ARI
5001 Eisenhower Avenue
Alexandria, VA 22333

BAGGETT PATRICIA
Dr. Patricia Baggett
School of Education
610 E. University, Rm 1302D
University of Michigan
Ann Arbor, MI 48109-1259

BAKER EVA L
Dr. Eva L. Baker
UCLA Center for the Study
of Evaluation
145 Moore Hall
University of California
Los Angeles, CA 90024

BAKER MERYL
Dr. Meryl S. Baker
Navy Personnel R&D Center
San Diego, CA 92152-6800

BAMBER DONALD E
Dr. Donald E. Bamber
Code 41
Navy Personnel R & D Center
San Diego, CA 92152-6800

BART WILLIAM M
Dr. William M. Bart
University of Minnesota
Dept. of Educ. Psychology
330 Burton Hall
178 Pillsbury Dr., S.E.
Minneapolis, MN 55455

BEJAR ISAAC
Dr. Isaac Bejar
Mail Stop: 10-R
Educational Testing Service
Rosedale Road
Princeton, NJ 08541

BLACK JOHN
Dr. John Black
Teachers College, Box 8
Columbia University
525 West 120th Street
New York, NY 10027

BOCK R DARRELL
Dr. R. Darrell Bock
University of Chicago
NORC
6030 South Ellis
Chicago, IL 60637

BONAR JEFF
Dr. Jeff Bonar
Learning R&D Center
University of Pittsburgh
Pittsburgh, PA 15260

BREAUX ROBERT
Dr. Robert Breaux
Code 7B
Naval Training Systems Center
Orlando, FL 32813-7100

BROWN ANN
Dr. Ann Brown
Center for the Study of Reading
University of Illinois
51 Gerty Drive
Champaign, IL 61280

BROWN JOHN S
Dr. John S. Brown
XEROX Palo Alto Research
Center
3333 Coyote Road
Palo Alto, CA 94304

BRUER JOHN T
Dr. John T. Bruer
James S. McDonnell Foundation
Suite 1610
1034 South Brentwood Blvd.
St. Louis, MO 63117

BUCHANAN BRUCE
Dr. Bruce Buchanan
Computer Science Department
Stanford University
Stanford, CA 94305

BURNS HUGH
LT COL Hugh Burns
AFHRL/IDI
Brooks AFB, TX 78235

CAREY SUSAN
Dr. Susan Carey
Department of Cognitive
and Neural Science
MIT
Cambridge, MA 02139

CARPENTER PAT
Dr. Pat Carpenter
Carnegie-Mellon University
Department of Psychology
Pittsburgh, PA 15213

CHARNEY DAVIDA
Dr. Davida Charney
English Department
Penn State University
University Park, PA 16802

CHI MICHELENE
Dr. Michelene Chi
Learning R & D Center
University of Pittsburgh
3939 O'Hara Street
Pittsburgh, PA 15260

CLANCEY WILLIAM
Dr. William Clancey
Institute for Research
on Learning
3333 Coyote Hill Road
Palo Alto, CA 94304

CNET N-5
Assistant Chief of Staff
for Research, Development,
Test, and Evaluation
Naval Education and
Training Command (N-5)
NAS Pensacola, FL 32508

COLLINS ALLAN M
Dr. Allan M. Collins
Bolt Beranek & Newman, Inc.
10 Moulton Street
Cambridge, MA 02238

COLLYER STANLEY
Dr. Stanley Collyer
Office of Naval Technology
Code 222
800 N. Quincy Street
Arlington, VA 22217-5000

CORBETT ALBERT T
Dr. Albert T. Corbett
Department of Psychology
Carnegie-Mellon University
Pittsburgh, PA 15213

CTB/MCGRAW-HILL LIBRARY
CTB/McGraw-Hill Library
2500 Garden Road
Monterey, CA 93940

CZICHON CARY
Dr. Cary Czichon
Intelligent Instructional Systems
Texas Instruments AI Lab
P.O. Box 660246
Dallas, TX 75266

DALLMAN BRIAN
Brian Dallman
Training Technology Branch
3400 TCHTW/TTGXC
Lowry AFB, CO 80230-5000

DERRY SHARON
Dr. Sharon Derry
Florida State University
Department of Psychology
Tallahassee, FL 32306

DTIC
Defense Technical
Information Center
Cameron Station, Bldg 5
Alexandria, VA 22314
Attn: TC
(12 Copies)

DURAN RICHARD
Dr. Richard Duran
Graduate School of Education
University of California
Santa Barbara, CA 93106

ELLIS JOHN
Dr. John Ellis
Navy Personnel R&D Center
Code 51
San Diego, CA 92252

EMBRETSON SUSAN
Dr. Susan Embretson
University of Kansas
Psychology Department
426 Fraser
Lawrence, KS 66045

ERIC
ERIC Facility-Acquisitions
4350 East-West Hwy., Suite 1100
Bethesda, MD 20814-4475

FARR MARSHALL J
Dr. Marshall J. Farr, Consultant
Cognitive & Instructional Sciences
2520 North Vernon Street
Arlington, VA 22207

FEDERICO PAT-ANTHONY
Dr. P-A. Federico
Code 51
NPRDC
San Diego, CA 92152-6800

FELTOVICH PAUL
Dr. Paul Feltovich
Southern Illinois University
School of Medicine
Medical Education Department
P.O. Box 3926
Springfield, IL 62708

FEURZEIG WALLACE
Mr. Wallace Feurzeig
Educational Technology
Bolt Beranek & Newman
10 Moulton St.
Cambridge, MA 02238

FLOWER LINDA
Dr. Linda Flower
Carnegie-Mellon University
Department of English
Pittsburgh, PA 15213

FORBUS KENNETH
Dr. Kenneth D. Forbus
University of Illinois
Department of Computer Science
1304 West Springfield Avenue
Urbana, IL 61801

FOX BARBARA A
Dr. Barbara A. Fox
University of Colorado
Department of Linguistics
Boulder, CO 80309

FREDERIKSEN CARL
Dr. Carl H. Frederiksen
Dept. of Educational Psychology
McGill University
3700 McTavish Street
Montreal, Quebec
CANADA H3A 1Y2

FREDERIKSEN JOHN R
Dr. John R. Frederiksen
BBN Laboratories
10 Moulton Street
Cambridge, MA 02238

FREDERIKSEN NORMAN
Dr. Norman Frederiksen
Educational Testing Service
(05-R)
Princeton, NJ 08541

GENTNER DEDRE
Dr. Dedre Gentner
University of Illinois
Department of Psychology
603 E. Daniel St.
Champaign, IL 61820

GIBBONS ROBERT D
Dr. Robert D. Gibbons
Illinois State Psychiatric Inst.
Rm 529W
1601 W. Taylor Street
Chicago, IL 60612

GINSBURG HERBERT
Dr. Herbert Ginsburg
Box 184
Teachers College
Columbia University
525 West 121st Street
New York, NY 10027

GLASER ROBERT
Dr. Robert Glaser
Learning Research
& Development Center
University of Pittsburgh
3939 O'Hara Street
Pittsburgh, PA 15260

GLENBERG ARTHUR M
Dr. Arthur M. Glenberg
University of Wisconsin
W. J. Brogden Psychology Bldg.
1202 W. Johnson Street
Madison, WI 53706

GOLDMAN SUSAN
Dr. Susan R. Goldman
Dept. of Education
University of California
Santa Barbara, CA 93106

GOTT SHERRIE
Dr. Sherrie Gott
AFHRL/MOMJ
Brooks AFB, TX 78235-5601

GOVINDARAJ T
Dr. T. Govindaraj
Georgia Institute of
Technology
School of Industrial
and Systems Engineering
Atlanta, GA 30332-0205

GRAY WAYNE
Dr. Wayne Gray
Army Research Institute
5001 Eisenhower Avenue
Alexandria, VA 22333

GREEN BERT
Dr. Bert Green
Johns Hopkins University
Department of Psychology
Charles & 34th Street
Baltimore, MD 21218

GREENO JAMES G
Dr. James G. Greeno
School of Education
Stanford University
Room 311
Stanford, CA 94305

HAERTEL EDWARD
Prof. Edward Haertel
School of Education
Stanford University
Stanford, CA 94305

HAMBLETON RONALD K
Dr. Ronald K. Hambleton
University of Massachusetts
Laboratory of Psychometric
and Evaluative Research
Hills South, Room 152
Amherst, MA 01003

HANNAPEL RAY
Dr. Ray Hannapel
Scientific and Engineering
Personnel and Education
National Science Foundation
Washington, DC 20550

HARVEY WAYNE
Dr. Wayne Harvey
Center for Learning Technology
Education Development Center
55 Chapel Street
Newton, MA 02160

HAYES JOHN R
Dr. John R. Hayes
Carnegie-Mellon University
Department of Psychology
Schenley Park
Pittsburgh, PA 15213

HOLLAND MELISSA
Dr. Melissa Holland
Army Research Institute for the
Behavioral and Social Sciences
5001 Eisenhower Avenue
Alexandria, VA 22333

HOLYOAK KEITH
Dr. Keith Holyoak
Department of Psychology
University of California
Los Angeles, CA 90024

HUTCHINS ED
Dr. Ed Hutchins
Intelligent Systems Group
Institute for
Cognitive Science (C-015)
UCSD
La Jolla, CA 92093

JACKSON JANET
Dr. Janet Jackson
Rijksuniversiteit Groningen
Biologisch Centrum, Vleugel D
Kerklaan 30, 9751 NN Haren
The NETHERLANDS

JANNARONE ROBERT
Dr. Robert Jannarone
Elec. and Computer Eng. Dept.
University of South Carolina
Columbia, SC 29208

JANVIER CLAUDE
Dr. Claude Janvier
Universite' du Quebec a Montreal
P.O. Box 8888, succ: A"
Montreal, Quebec H3C 3P8
CANADA

JEFFRIES ROBIN
Dr. Robin Jeffries
Hewlett-Packard Laboratories, 3L
P.O. Box 10490
Palo Alto, CA 94303-0971

JONES DOUGLAS H
Dr. Douglas H. Jones
Thatcher Jones Associates
P.O. Box 6640
10 Trafalgar Court
Lawrenceville, NJ 08648

JUST MARCEL
Dr. Marcel Just
Carnegie-Mellon University
Department of Psychology
Schenley Park
Pittsburgh, PA 15213

KATZ MILTON S
Dr. Milton S. Katz
European Science Coordination
Office
U.S. Army Research Institute
Box 65
FPO New York 09510-1500

KELLOGG WENDY
Dr. Wendy Kellogg
IBM T. J. Watson Research Ctr.
P.O. Box 704
Yorktown Heights, NY 10598

KIBLER DENNIS
Dr. Dennis Kibler
University of California
Department of Information
and Computer Science
Irvine, CA 92717

KIERAS DAVID
Dr. David Kieras
Technical Communication Program
TIDAL Bldg., 2360 Bonisteel Blvd.
University of Michigan
Ann Arbor, MI 48109-2108

KINCAID J PETER
Dr. J. Peter Kincaid
Army Research Institute
Orlando Field Unit
c/o PM TRADE-E
Orlando, FL 32813

KINTSCH WALTER
Dr. Walter Kintsch
Department of Psychology
University of Colorado
Boulder, CO 80309-0345

KLAHR DAVID
Dr. David Klahr
Carnegie-Mellon University
Department of Psychology
Schenley Park
Pittsburgh, PA 15213

KOTOVSKY KENNETH
Dr. Kenneth Kotovsky
Community College of
Allegheny County
808 Ridge Avenue
Pittsburgh, PA 15212

KRANTZ DAVID H
Dr. David H. Krantz
Department of Psychology
Columbia University
406 Schermerhorn Hall
New York, NY 10027

KYLLONEN PATRICK
Dr. Patrick Kyllonen
Institute for Behavioral
Research
Graduate Studies Bldg.
University of Georgia
Athens, GA 30602

LANGLEY PAT
Dr. Pat Langley
University of California
Department of Information
and Computer Science
Irvine, CA 92717

LARKIN JILL
Dr. Jill Larkin
Carnegie-Mellon University
Department of Psychology
Pittsburgh, PA 15213

LAVE JEAN
Dr. Jean Lave
Institute for Research
on Learning
3333 Coyote Hill Road
Palo Alto, CA 92304

LAWLER ROBERT
Dr. Robert W. Lawler
Matthews 118
Purdue University
West Lafayette, IN 47907

LESGOLD ALAN
Dr. Alan M. Lesgold
Learning R&D Center
University of Pittsburgh
Pittsburgh, PA 15260

LEVIN JAMES
Dr. Jim Levin
Department of
Educational Psychology
210 Education Building
1310 South Sixth Street
Champaign, IL 61820-6990

LEVINE JOHN
Dr. John Levine
Learning R&D Center
University of Pittsburgh
Pittsburgh, PA 15260

LEVINE MICHAEL
Dr. Michael Levine
Educational Psychology
210 Education Bldg.
University of Illinois
Champaign, IL 61801

LEWIS CLAYTON
Dr. Clayton Lewis
University of Colorado
Department of Computer Science
Campus Box 430
Boulder, CO 80309

LEWIS MATT
Matt Lewis
Department of Psychology
Carnegie-Mellon University
Pittsburgh, PA 15213

LIBRARY NTSC
Library
Naval Training Systems Center
Orlando, FL 32813

LIBRARY NWC
Library
Naval War College
Newport, RI 02940

LIBRARY OF CONGRESS
Science and Technology Division
Library of Congress
Washington, DC 20540

LINN MARCIA C
Dr. Marcia C. Linn
Graduate School
of Education, EMST
Tolman Hall
University of California
Berkeley, CA 94720

LINN ROBERT L
Dr. Robert L. Linn
Campus Box 249
University of Colorado
Boulder, CO 80309-0249

MALOY WILLIAM L
Dr. William L. Maloy
Naval Education and Training
Program Support Activity
Code 047
Building 2435
Pensacola, FL 32509-5000

MARSHALL SANDRA P
Dr. Sandra P. Marshall
Dept. of Psychology
San Diego State University
San Diego, CA 92182

MAYER RICHARD
Dr. Richard E. Mayer
Department of Psychology
University of California
Santa Barbara, CA 93106

MCBRIDE JAMES R
Dr. James R. McBride
The Psychological Corporation
1250 Sixth Avenue
San Diego, CA 92101

MCDONALD BARBARA
Dr. Barbara McDonald
Navy Personnel R&D Center
San Diego, CA 92152-6800

MCLACHLAN JOSEPH C
Dr. Joseph C. McLachlan
Code 52
Navy Personnel R&D Center
San Diego, CA 92152-6800

MCMICHAEL JAMES
Dr. James McMichael
Technical Director
Navy Personnel R&D Center
San Diego, CA 92152-6800

MEANS BARBARA
Dr. Barbara Means
SRI International
333 Ravenswood Avenue
Menlo Park, CA 94025

MESTRE JOSE
Dr. Jose Mestre
Department of Physics
Hasbrouck Laboratory
University of Massachusetts
Amherst, MA 01003

MILLER GEORGE A
Dr. George A. Miller
Dept. of Psychology
Green Hall
Princeton University
Princeton, NJ 08540

MISLEVY ROBERT
Dr. Robert Mislevy
Educational Testing Service
Princeton, NJ 08541

MOLNAR ANDREW R
Dr. Andrew R. Molnar
Applic. of Advanced Technology
Science and Engr. Education
National Science Foundation
Washington, DC 20550

MONTAGUE WILLIAM
Dr. William Montague
NPRDC Code 13
San Diego, CA 92152-6800

MUNRO ALLEN
Dr. Allen Munro
Behavioral Technology
Laboratories - USC
1845 S. Elena Ave., 4th Floor
Redondo Beach, CA 90277

NISBETT RICHARD E
Dr. Richard E. Nisbett
University of Michigan
Institute for Social Research
Room 5261
Ann Arbor, MI 48109

NORMAN DONALD A
Dr. Donald A. Norman
C-015
Institute for Cognitive Science
University of California
La Jolla, CA 92093

NPRDC 01A
Deputy Technical Director
NPRDC Code 01A
San Diego, CA 92152-6800

NPRDC 05
Director, Training Laboratory,
NPRDC (Code 05)
San Diego, CA 92152-6800

NPRDC 06
Director, Manpower and Personnel
Laboratory,
NPRDC (Code 06)
San Diego, CA 92152-6800

NPRDC 07
Director, Human Factors
& Organizational Systems Lab,
NPRDC (Code 07)
San Diego, CA 92152-6800

NPRDC LIBRARY
Library, NPRDC
Code P201L
San Diego, CA 92152-6800

NPRDC TECHNICAL DIRECTOR
Technical Director
Navy Personnel R&D Center
San Diego, CA 92152-6800

NRL CODE 2627
Commanding Officer,
Naval Research Laboratory
Code 2627
Washington, DC 20390

O'NEIL HARRY F
Dr. Harold F. O'Neil, Jr.
School of Education - WPH 801
Department of Educational
Psychology & Technology
University of Southern California
Los Angeles, CA 90089-0031

OHLSSON STELLAN
Dr. Stellan Ohlsson
Learning R & D Center
University of Pittsburgh
Pittsburgh, PA 15260

ONR CODE 1142
Office of Naval Research,
Code 1142
800 N. Quincy St.
Arlington, VA 22217-5000

ONR CODE 1142BI
Office of Naval Research,
Code 1142BI
800 N. Quincy Street
Arlington, VA 22217-5000

ONR CODE 1142CS
Office of Naval Research,
Code 1142CS
800 N. Quincy Street
Arlington, VA 22217-5000
(6 Copies)

ONR CODE 1142PS
Office of Naval Research,
Code 1142PS
800 N. Quincy Street
Arlington, VA 22217-5000

ONR LONDON
Psychologist
Office of Naval Research
Branch Office, London
Box 39
FPO New York, NY 09510

ONR MARINE CORPS
Special Assistant for Marine
Corps Matters,
ONR Code 00MC
800 N. Quincy St.
Arlington, VA 22217-5000

ORASANU JUDITH
Dr. Judith Orasanu
Basic Research Office
Army Research Institute
5001 Eisenhower Avenue
Alexandria, VA 22333

PAULSON JAMES

Dr. James Paulson
Department of Psychology
Portland State University
P.O. Box 751
Portland, OR 97207

PEARCE DOUGLAS
Dr. Douglas Pearce
1133 Sheppard W
Box 2000
Downsview, Ontario
CANADA M3M 3B9

PENTAGON TRAINING & PERSONNEL TECHNOLOGY
Military Assistant for Training and
Personnel Technology,
OUSD (R & E)
Room 3D129, The Pentagon
Washington, DC 20301-3080

PEREZ RAY S
Dr. Ray S. Perez
ARI (PERI-II)
5001 Eisenhower Avenue
Alexandria, VA 22333

PERKINS DAVID N
Dr. David N. Perkins
Project Zero
Harvard Graduate School
of Education
7 Appian Way
Cambridge, MA 02138

PERRY NANCY N
Dr. Nancy N. Perry
Naval Education and Training
Program Support Activity
Code-047
Building 2435
Pensacola, FL 32509-5000

PIROLLI PETER
Dr. Peter Pirolli
School of Education
University of California
Berkeley, CA 94720

PLOMP TJEERD
Dr. Tjeerd Plomp
Twente University of Technology
Department of Education
P.O. Box 217
7500 AE ENSCHEDE
THE NETHERLANDS

POLSON MARTHA
Dr. Martha Polson
Department of Psychology
University of Colorado
Boulder, CO 80309-0345

PSOTKA JOSEPH
Dr. Joseph Psotka
ATTN: PERI-IC
Army Research Institute
5001 Eisenhower Ave.
Alexandria, VA 22333-5600

RECKASE MARK D
Dr. Mark D. Reckase
ACT
P. O. Box 168
Iowa City, IA 52243

REDER STEVE
Dr. Steve Reder
Northwest Regional
Educational Laboratory
400 Lindsay Bldg.
710 S.W. Second Ave.
Portland, OR 97204

REIF FRED
Dr. Fred Reif
Physics Department
University of California
Berkeley, CA 94720

RESNICK LAUREN
Dr. Lauren Resnick
Learning R & D Center
University of Pittsburgh
3939 O'Hara Street
Pittsburgh, PA 15213

RICHARDSON J JEFFREY
Dr. J. Jeffrey Richardson
Center for Applied AI
College of Business
University of Colorado
Boulder, CO 80309-0419

RISSLAND EDWINA L
Dr. Edwina L. Rissland
Dept. of Computer and
Information Science
University of Massachusetts
Amherst, MA 01003

ROBERTS LINDA G
Dr. Linda G. Roberts
Science, Education, and
Transportation Program
Office of Technology Assessment
Congress of the United States
Washington, DC 20510

RUBIN DONALD
Dr. Donald Rubin
Statistics Department
Science Center, Room 608
1 Oxford Street
Harvard University
Cambridge, MA 02138

SAMEJIMA FUMIKO
Dr. Fumiko Samejima
Department of Psychology
University of Tennessee
310B Austin Peay Bldg.
Knoxville, TN 37916-0900

SCHANK ROGER
Dr. Roger Schank
Yale University
Computer Science Department
P.O. Box 2158
New Haven, CT 06520

SCHOENFELD ALAN H
Dr. Alan H. Schoenfeld
University of California
Department of Education
Berkeley, CA 94720

SCHOFIELD JANET W
Dr. Janet W. Schofield
816 LRDC Building
University of Pittsburgh
3939 O'Hara Street
Pittsburgh, PA 15260

SEGAL JUDITH W
Dr. Judith W. Segal
OERI
555 New Jersey Ave., NW
Washington, DC 20208

SEIFERT COLLEEN M
Dr. Colleen M. Seifert
Institute for Cognitive Science
Mail Code C-015
University of California, San Diego
La Jolla, CA 92093

SHULMAN LEE S
Dr. Lee S. Shulman
School of Education
507 Ceras
Stanford University
Stanford, CA 94305-3084

SIEGLER ROBERT S
Dr. Robert S. Siegler
Carnegie-Mellon University
Department of Psychology
Schenley Park
Pittsburgh, PA 15213

SILVER EDWARD
Dr. Edward Silver
LRDC
University of Pittsburgh
3939 O'Hara Street
Pittsburgh, PA 15260

SIMON HERBERT A
Dr. Herbert A. Simon
Department of Psychology
Carnegie-Mellon University
Schenley Park
Pittsburgh, PA 15213

SLEEMAN DEREK
Dr. Derek Sleeman
Computing Science Department
King's College
Old Aberdeen AB9 2UB
Scotland
UNITED KINGDOM

SNOW RICHARD
Dr. Richard E. Snow
School of Education
Stanford University
Stanford, CA 94305

SOLOWAY ELLIOT
Dr. Elliot Soloway
Yale University
Computer Science Department
P.O. Box 2158
New Haven, CT 06520

SORENSEN RICHARD C
Dr. Richard C. Sorensen
Navy Personnel R&D Center
San Diego, CA 92152-6800

SPECKMAN PAUL
Dr. Paul Speckman
University of Missouri
Department of Statistics
Columbia, MO 65201

STEARNS MARIAN
Dr. Marian Stearns
SRI International
333 Ravenswood Ave.
Room B-5124
Menlo Park, CA 94025

STERNBERG ROBERT J
Dr. Robert J. Sternberg
Department of Psychology
Yale University
Box 11A, Yale Station
New Haven, CT 06520

STEVENS ALBERT
Dr. Albert Stevens
Bolt Beranek & Newman, Inc.
10 Moulton St.
Cambridge, MA 02238

STICHT THOMAS
Dr. Thomas Sticht
Applied Behavioral and
Cognitive Sciences, Inc.
P.O. Box 6640
San Diego, CA 92106

SUPPES PATRICK
Dr. Patrick Suppes
Stanford University
Institute for Mathematical
Studies in the Social Sciences
Stanford, CA 94305-4115

SWAMINATHAN HARIHARAN
Dr. Hariharan Swaminathan
Laboratory of Psychometric and
Evaluation Research
School of Education
University of Massachusetts
Amherst, MA 01003

SYMPSON BRAD
Mr. Brad Sympson
Navy Personnel R&D Center
Code-62
San Diego, CA 92152-6800

TANGNEY JOHN
Dr. John Tangney
AFOSR/NL, Bldg. 410
Bolling AFB, DC 20332-6448

TATSUOKA KIKUMI
Dr. Kikumi Tatsuoka
CERL
252 Engineering Research
Laboratory
103 S. Mathews Avenue
Urbana, IL 61801

TAYLOR M MARTIN
Dr. M. Martin Taylor
DCIEM
Box 2000
Downsview, Ontario
CANADA M3M 3B9

THORNDYKE PERRY W
Dr. Perry W. Thorndyke
FMC Corporation
Central Engineering Labs
1205 Coleman Avenue, Box 580
Santa Clara, CA 95052

TOWNE DOUGLAS
Dr. Douglas Towne
Behavioral Technology Labs
University of Southern California
1845 S. Elena Ave.
Redondo Beach, CA 90277

TSUTAKAWA ROBERT
Dr. Robert Tsutakawa
University of Missouri
Department of Statistics
222 Math. Sciences Bldg.
Columbia, MO 65211

TWOHIG PAUL T
Dr. Paul T. Twohig
Army Research Institute
5001 Eisenhower Avenue
ATTN: PERI-RL
Alexandria, VA 22333-5600

TYER ZITA E
Dr. Zita E. Tyer
Department of Psychology
George Mason University
4400 University Drive
Fairfax, VA 22030

USMC HQ
Headquarters, U. S. Marine Corps
Code MPI-20
Washington, DC 20380

VALE DAVID
Dr. David Vale
Assessment Systems Corp.
2233 University Avenue
Suite 440
St. Paul, MN 55114

VAN LEHN KURT
Dr. Kurt Van Lehn
Department of Psychology
Carnegie-Mellon University
Schenley Park
Pittsburgh, PA 15213

WANG MING-MEI
Dr. Ming-Mei Wang
Lindquist Center
for Measurement
University of Iowa
Iowa City, IA 52242

WARREN BETH
Dr. Beth Warren
BBN Laboratories, Inc.
10 Moulton Street
Cambridge, MA 02238

WHITE BARBARA
Dr. Barbara White
BBN Laboratories
10 Moulton Street
Cambridge, MA 02238

WING HILDA
Dr. Hilda Wing
NRC MH-176
2101 Constitution Ave.
Washington, DC 20418

WISHER ROBERT A
Dr. Robert A. Wisher
U.S. Army Institute for the
Behavioral and Social Sciences
5001 Eisenhower Avenue
Alexandria, VA 22333-5600

WISKOFF MARTIN F
Dr. Martin F. Wiskoff
Defense Manpower Data Center
550 Camino El Estero
Suite 200
Monterey, CA 93943-3231

WITTROCK MERLIN C
Dr. Merlin C. Wittrock
Graduate School of Education
UCLA
Los Angeles, CA 90024

WOLFE JOHN H
Mr. John H. Wolfe
Navy Personnel R&D Center
San Diego, CA 92152-6800

WONG GEORGE
Dr. George Wong
Biostatistics Laboratory
Memorial Sloan-Kettering
Cancer Center
1275 York Avenue
New York, NY 10021

WULFECK WALLACE
Dr. Wallace Wulfeck, III
Navy Personnel R&D Center
Code 51
San Diego, CA 92152-6800

YAZDANI MASOUD
Dr. Masoud Yazdani
Dept. of Computer Science
University of Exeter
Prince of Wales Road
Exeter EX44PT
ENGLAND

YOUNG DR JOSEPH L
Dr. Joseph L. Young
National Science Foundation
Room 320
1800 G Street, N.W.
Washington, DC 20550